# CRANFIELD UNIVERSITY

## COLLEGE OF AERONAUTICS

## ENGINEERING DOCTORATE THESIS

**1994-1998**

**R.M.TREBILCOCK**

# Intelligent Link Between Design, CAD and Structural Analysis

1998

**Academic Supervisor**: Professor A.J.Morris
College of Aeronautics, Cranfield University

**Industrial Supervisor**: Mr. D.Thompson
British Aerospace
Military Aircraft and Aerostructures

**Management Supervisor**: Mr. J.C.Baker
Cranfield School of Management,
Cranfield University

**To my Mother and Father**

# Abstract

This study outlines the requirement for a company to be able to manage its intellectual capital. On the basis of this requirement this study presents a new design methodology based around the requirements of the military aircraft industry. It tackles the difficult management problem of capturing, storing and re-using valuable company product knowledge. The detailed research documented in this thesis focuses on the conceptual design area of this methodology. Work in this area has resulted in the development of a further methodology for the conceptual design arena. This methodology is called the intelligent conceptual engineering system (ICES). The ICES methodology embraces the artificial intelligence disciplines of knowledge-based systems and case-based reasoning. Through the evolutionary development of the ICES methodology a significant contribution to knowledge has been made in three areas. Firstly, this study introduces a new method of assigning justifiable numerical weights to design drivers acting on the design process. Secondly, the work introduces the novel concept of using secondary rules in the knowledge-based system so a 'best structure' can be derived from the manufacturing and structures perspectives. Finally, this work adds a new concept to case-based reasoning called the 'jury technique'. These concepts, developed to support the ICES methodology have been placed in a prototype design decision support tool.

## Acknowledgements

There are many people who over the duration of this study have in one way or another contributed to the successful conclusion of this research project. I would like to take this opportunity to thank them.

I would like in particular to thank Professor Alan Morris, my academic supervisor, for his assistance throughout the duration of this research project. I always enjoyed our discussions and his input was enlightening and often invaluable.

Special thanks to Dave Thompson who as the industrial supervisor for this research project was superb. His input at critical moments throughout the duration of this study helped to ensure it reached a successful conclusion. I would also like to take this opportunity to express my appreciation for all the assistance I was given by staff at British Aerospace, Warton. This project could never have been completed without their readily given help and guidance.

I would like to express my thanks to John Baker for his input as my management supervisor.

Listed below are people who I wish to extend special thanks to for there assistance over the duration of this study.
Pauline Forshaw
Rade Vignjevic
Les Oswald
Jody Hagins

# Contents

# List of Figures

# Chapter 1

# Introduction

The current costs of developing new military aircraft are becoming prohibitive. Gardener has commented, "The business challenges faced by British Aerospace, Military Aircraft and Aerostructures are high. We are an advanced engineering company in a global market, and are driven by the need to reduce cost and programme development time scales, whilst increasing design quality and enabling innovation".[1] Further, it was reported by the Joint Strike Fighter (JSF) programme that if the costs of developing tactical aircraft continues to increase at its current rate by the end of the next century costs associated with developing new tactical aircraft will absorb the entire defence budget for the United States of America.[2,3] The problem of spiralling military aircraft development costs is so serious that multi-national initiatives have been instigated to address the problem, of which the JSF programme is at the time of writing, possibly the most high profile. The goal is to develop a family of tactical aircraft which meet the next generation strike mission needs of the US Navy, Marines, Air Force and Allied Forces. The corner stone of this programme is affordability making the JSF programme a leader in affordability activities such as the Lean Aircraft Initiative (LAI).[4]

All companies world wide involved in the development of new military aircraft recognise the need to reduce development costs and increase company innovative capacity. Clearly, these twin goals are difficult to reconcile. However, a key factor which will facilitate a company's capability to make progress towards some degree of reconciliation is an appreciation of the importance of intellectual capital within the organisation and the need to maximise its use. Increasingly, company knowledge resources will provide the principle source of sustainable competitive advantage.[5]

In the process of delivering new products to market in ever reducing time scales, time and knowledge are inextricably linked as two parts of one continuum. Knowledge requires time to accumulate and structure and as time passes opportunities disappear and new issues, with new requirements for yet more knowledge, arise.[6] This requires a company putting in place a capability to facilitate the capture and storage of its product knowledge and provide the means to deliver this knowledge to the points-of-need more efficiently and effectively than ever before if the desired reduced time scales are to be met.

Having briefly identified the significant operating pressures present in today's military aircraft industry this thesis presents a new design methodology for British Aerospace, Military Aircraft and Aerostructures (BAe, MA&A). The underlying intention throughout the development of the methodology has been to incorporate the artificial intelligence (AI) technology necessary to facilitate the capture of product knowledge and deliver that knowledge to the points of need.

This thesis is composed of eleven chapters, each of which addresses a specific area of the study undertaken, such that, the objectives of the research project are met

precisely and in a logical manner. In order to provide the reader with a guide or 'roadmap' to this thesis the contents of each chapter are now briefly outlined:

**Chapter 1** continues by discussing the importance of intellectual capital in today's manufacturing industry of which the military aircraft industry is a small but significant part; the chapter discusses the strategic importance of intellectual capital and the business needs for capturing product knowledge. This is followed by an overview of the requirements of a corporate knowledge strategy. The chapter then focuses on BAe MA&A strategic and operational view of knowledge management. Here focus is placed on the issues that confront BAe MA&A, with respect to both ensuring that existing knowledge is retained in the company, and also to provide the means to draw new knowledge into the company in an efficient and effective manner i.e., such that it can be employed within current projects.

Having highlighted the issues that confront BAe MA&A with respect to knowledge management, the remainder of this chapter provides the reader with an overview of the new design methodology. In the context of this thesis, the aim of this overview is to provide the reader with a 'picture' of the methodology before leading on to discuss the technical detail and concepts which support the operation of the methodology.

**Chapter 2** consists of a literature review, and places the work documented in this thesis in the context of previous research performed in the area. The chapter highlights the significant differences between this study and what has been researched previously.

**Chapter 3** discusses BAe MA&A's current design methodology as viewed from the Structures Unit. In terms of knowledge management, this chapter identifies the types of knowledge present in the current design methodology and how it is limited with respect to being channelled or directed in the appropriate areas to facilitate the capture and subsequent re-use of product knowledge. In addition, this chapter highlights those areas within the current methodology where AI technology could be employed to facilitate the capture of product knowledge.

**Chapter 4** discusses the AI disciplines of case-based reasoning (CBR) and knowledge-based systems (KBS). These being the disciplines selected for use within this study. In addition, this chapter discusses other AI disciplines that where considered for use within this study and the reasons for their eventual rejection.

**Chapter 5** discusses in detail the new design methodology, first briefly introduced in Chapter 1. The chapter focuses on the placement of the AI technology within the methodology. As the discussion works through the methodology in a logical manner attention is paid to the particular reasons for the placement of AI technology in the areas discussed. Towards the end of the chapter it is made clear to the reader that the full development of the methodology is not possible within the time scale allotted for this study. Further, a reasoned argument is presented why the detailed research of this project focused on the conceptual design arena. This being a small but significant segment of the methodology presented. Finally, the chapter discusses the need validate the concepts presented in the conceptual design arena.

**Chapter 6** discusses the background knowledge in design. In particular, focus is placed on the role of case-based reasoning (CBR) in the aircraft design process. The chapter also discusses the role of research and development (R&D) in the design process.

**Chapter 7** presents the technical detail that supports the Intelligent Conceptual Engineering System (ICES) methodology. This is the principle chapter of this thesis. *This chapter documents the contribution to knowledge made during this research project. The contribution to knowledge documented in this chapter lies in two areas. Firstly, the design driver ranking technique which assigns justifiable numerical weightings to design drivers acting on the design process in conjunction with the approach employed to ensure consensus between a manufacturing rule base and a structures rule base with respect to deriving a 'best structure'; specifically, this approach employs the concept of utilising secondary rules. Secondly, this chapter introduces a highly novel new concept to case base reasoning called the 'jury technique'.*

**Chapter 8** discusses the issues that needed to be addressed in order to develop a software implementation of ICES. This chapter outlines the software and hardware selection process which at first glance may appear a straight forward procedure but in fact evolved in conjunction with the development of the ICES.

**Chapter 9** describes in detail the software implementation of the ICES methodology. The chapter starts with an overview of the software. This is followed by a logical step-by-step description of the ICES prototype functionality. The chapter concludes with a step-through example run of the ICES KBS and case base implementation.

**Chapter 10** analyses the results output by the ICES prototype. In particular, attention is paid to the quality of the results. This analysis includes a discussion of the validity of what the system has actually done with respect the results presented.

**Chapter 11** contains the research project conclusions. In this chapter the new methodology and in particular ICES are viewed from a global and systems perspective. Here the work presented in this thesis will be analysed in the context of whether it represents a move towards a system that truly can facilitate the capture and subsequent utilisation of company product knowledge. In addition, the possible ways in which the concepts presented in this study may be taken further by BAe MA&A are discussed.

## 1.1. The Strategic Importance of Intellectual Capital

In the business world, the dawning of a new century seems to coincide with a fundamental reappraisal of what is really important for business success. At the turn of the nineteenth century, the pursuit of property by the Empire capitalists gave way to a new emphasis on the wealth in the early mass production industries. Now as businesses approach the new millennium it seems that the acquisition and management of this financial capital is no longer their primary concern. Rather, the

twenty-first century organisation will be focused on its *intellectual* capital, as it will increasingly be the case that the knowledge resources of a firm will be its source of competitive advantage.[7] The knowledge resources or *assets* of a company is that knowledge regarding markets, products, technologies and organisation, that a business owns or needs to own and which enable its business processes to generate profits. As the importance of knowledge assets become appreciated so has the need to manage them; hence the importance placed on knowledge management. Knowledge management involves the identification and analysis of available and required knowledge assets and knowledge related processes, and the subsequent planning and control of actions to develop both assets and processes so as to fulfil organisational objectives. Whilst the importance of knowledge and its management has been understood by many firms in service industries for some time, there are relatively few manufacturers who have taken heed of the advice offered by leading academics such as Peter Drucker who has called the substitution of knowledge for manual effort as "the greatest change in the history of work".

What is different about the business environment today which requires such a shift in emphasis? Trends at organisational, industry and global levels appear to be towards increasing diversity and unpredictability, driven in part by socio-economic phenomena like "consumerism" leading to what has been called by some authors a "post-industrial" business environment.[8] The mass production paradigm which has been dominant through the twentieth century is poorly suited to such a 'turbulent' climate, even with its more recent refinements such as lean production. This is because the underlying premise is that if producers supply good enough, cheap enough standard products and market them well enough, customers will buy them. Under this way of thinking, running a manufacturing business is essentially concerned with finance-related decisions such as break-even points, labour productivity and capital productivity. However, the contemporary marketplace wants customised products, innovative products, and products and services packaged into complete 'solutions'. This is creating what economists Piore and Sabel have called 'the second industrial divide' and is switching the attention of managers to new ideas such as Drucker's 'productivity of knowledge'.[9,10]

It is important to clarify what is meant by 'knowledge' in this context and an immediate distinction to be drawn is between data, information and knowledge:

- Data - symbols which have not yet been interpreted.
- Information - data which has been assigned a meaning and is always linked to specific situations.
- Knowledge - enables people to assign a meaning to data and so generate information

Hudson distinguishes between information and knowledge on two grounds[11]:

- That knowledge is processed information.
- That knowledge exists in the human mind, not in books or computers.

There are two generic types of knowledge:

- Explicit knowledge, which is formal and readily articulated

- Tacit which is informal, personal and not usually, or easily expressed. According to Nonaka tacit knowledge includes the 'technical-know-how' possessed by craftspeople and professionals.[12]

Thus, the intellectual capital which is comprised of this knowledge is not something which an organisation can easily lock away in a vault, unlike its financial predecessor. Most organisations have historically relied on knowledge being passed on from one generation of employees to the next and indeed was the basis for the traditional apprenticeship. However, in the era of massive downsizing, early retirement, and out-sourcing much knowledge-creating work to suppliers, how is this immensely valuable knowledge now being retained within manufacturing firms? This thesis in the context of presenting a new design methodology for BAe, MA&A looks at a new approach to achieving 'knowledge productivity' in a fundamental area. That is, the design of complex products. Before proceeding it is appropriate here to summarise the motivating factors for a company to manage its knowledge:

- The rate of innovation is rising.
- The size of the workforce is reducing.
- There are trends for employees to retire earlier.
- There is less time to acquire knowledge.
- Knowledge may be lost due to changes in company strategy.
- Increasing need to apply knowledge globally.

### 1.1.1. Business Needs for Capturing Product Knowledge

With the increasing globalisation of industry there is a requirement for companies to be able to apply their expertise in all theatres of business where ever they may be in the world. This is further compounded by the requirement for companies to cope with ever shifting markets, proliferating technologies, increased competition and a situation where products become obsolete almost overnight. Clearly, the combination of globalisation and ever increasing demands of the market put considerable demands on company expertise.

With these pressures on company expertise, companies must pay specific attention to the sources of knowledge that reside in the company and ensure they are nurtured and not squandered. All companies have to face the problem that they will lose highly skilled personnel from time-to-time for a variety of reasons e.g., retirement, a better job opportunity with another company, illness, pregnancy, or possibly forced redundancies due to the economic climate. However, when a company loses skilled staff not only do they lose the person, but more importantly to the company, they lose valuable product knowledge. This knowledge, which the person will usually have amassed over many years working for the company cannot be replaced as quickly as the vacant post can be filled. In many instances, certain aspects of the knowledge that a particular member of staff possessed will be lost for ever to the company. This is particularly true in the case where personnel have worked for the company all their working lives, prior to retirement. In such cases, the rules used, while working for the company, would be based on sound judgement but also on an acute appreciation of the company's products, rivals and overall position in the market place.

Clearly, most companies readily acknowledge the need to retain product knowledge within the company in spite of staff turnover. The methods available to company to store and access product knowledge vary considerably. However, to a large extent the effort to which a company will go to retain its product knowledge will depend on several factors.

The complexity of the product is one of the most important factors to effect knowledge capture decisions. For instance, if one compares an aircraft manufacturer with a company producing nuts, washers and screws, the product knowledge associated with an aircraft is vastly more complex. In addition, there is the requirement for the aircraft manufacturer not only to be able to readily access its existing product knowledge but provide the means whereby this knowledge can be retained and built upon. Thus, enabling the company to produce aircraft that will be obvious improvements on previous models. Conversely, the company producing nuts, washers and screws has very simple and stable products, and the product knowledge associated with nuts, washers and screws should be well within the grasp of the vast majority of engineers at all levels of experience. As such, the company need not go to the elaborate lengths to capture and retain product knowledge that the aircraft manufacture is obliged to.

The requirements relating to the capture and storage of product knowledge will to some extent depend on the size of the company. The more people who make a direct input into the design and manufacturing process, the more 'complex' the procedure for capturing and storing product knowledge. Conversely, in a one person business all the product knowledge is retained by the individual e.g., a blacksmith.

The capture and storage of product knowledge can be a costly and time consuming exercise. To a large extent, the time and money a company can commit to this exercise will be the limiting factor with respect to the methods eventually selected.

While companies may readily acknowledge the need to capture their product knowledge there is also a need to identify responsibility within the company for knowledge capture. In this context there are many factors that must be considered; should knowledge be a separate function within firms? Should we have knowledge departments? Whose management responsibility is the nurturing and management of knowledge in the organisation? Who is to make the strategic decisions on what knowledge needs to be generated, what knowledge needs to be codified? Is it a functional decision? Is it an operational decision? Who makes those decisions?[13] If these questions are not answered company product knowledge may continue to be lost or put at risk. In those companies that are actively looking at the capture and deployment of knowledge throughout the company, a common solution to the problem of responsibility for knowledge is the introduction of the post of corporate knowledge officer (CKO). The CKO is the person who 'stands' at the centre of the information flow and has the sense and skill to see that it is connected to all the decision makers who need it. Typically, CKOs have experience in systems operations, which teaches how various activities fit together. Therefore, they can help independent departments to become more collaborative.[14]

Knowledge as with all company resources needs to be managed at both strategic level and at the operational level. Strategic knowledge management takes a company wide view of how knowledge assets in the company should be managed and directed. Operational knowledge management relates to the individual functions within the company and how they implement the strategic knowledge decisions defined by senior management. The following section now discusses what is required of a corporate knowledge strategy.

## 1.2. The Requirements of a Corporate Knowledge Strategy

Techniques for strategic-level knowledge management can be divided into two areas; knowledge asset management and process management. With respect to knowledge asset management it first necessary to identify knowledge assets at a strategic-level. This is typically carried out by business analysts and managers. Existing or potential knowledge assets are modelled at an abstract-level. The model will be developed with respect to:

- The business strategy (mission and objectives)
- The company's core competencies
- Key organisation processes
- Key services and products
- The company's organisational structure

Having identified the company's knowledge assets it is then necessary to analysis them with respect to how they relate to the company's core competencies. Some of the knowledge assets will be standard, being common to most other comparable companies; others will be promising and in need of development and there will be those which are very competitive and relate strongly to the company's current perceived core competencies. In addition, the knowledge asset analysis may identify problems with key knowledge assets. Assets maybe vulnerable and at risk of being lost. A further problem could be that the knowledge asset analysis reveals shortfalls or shortages of key assets. A good knowledge asset analysis should be often looking to the future; being pro-active rather than reactive; looking at knowledge assets that need to be developed and considering how the flexibility and capacity to deal with any required change will be put in place. Commonly used representation and analysis tools that may be used to model and analyse knowledge are:

- Knowledge asset roadmaps
- SWOT analysis
- Value chain analysis
- Knowledge inventories
- Portfolio management

Knowledge asset roadmaps show dependencies between knowledge assets and products, services, projects, or processes which use the assets. They indicate how the use of a knowledge asset will change over time. A further use of knowledge asset roadmaps is that they can be used to identify key assets i.e., assets upon which many other things depend.[15]

SWOT analysis typically looks at the company's internal strengths and weaknesses and the opportunities and threats posed by the external environment. It

can be used for assessing organisational strategy and key knowledge assets. SWOT may assist management to identify knowledge assets that are under threat and provide a starting point for assessing proposed actions.

One way of gaining a deeper insight into buyers needs is through value chain analysis. The value chain breaks down the firm into its strategically relevant activities in order to understand the behaviour of costs and the existing or potential sources of differentiation. A firm gains competitive advantage by performing these strategically important activities more cheaply or better than its rivals.[16]

Knowledge inventories typically attempt to measure the value of knowledge assets.[17] The measuring of the value of knowledge assets is a difficult task. Two methods that may provide insight to the value of knowledge assets are:

- The use of accounting measures
- Balance scorecards.

With accounting measures, known knowledge assets are compared with the balance sheet. Here an attempt is made to attribute financial value to particular knowledge assets in the context of their individual impact on the balance sheet. Clearly, this not an exact science but it should provide management with a 'feel' for the value of individual knowledge assets.

Balance scorecards differ from accounting measures in that they assess the value of a knowledge assets from four different perspectives:

- The business perspective,
- The customer perspective,
- The business process perspective
- The organisational leaning perspective

The method of applying scores in these areas should be simple e.g., a good/bad scoring system is acceptable. From the business perspective, each knowledge asset should be assessed with respect to how it impacts on business growth, profit, and shareholder value. From the customer perspective it is necessary to consider how the quality of knowledge has had an impact on customers in terms of the quality of service. In addition, this analysis must also embrace the value and quality of the product(s) delivered as perceived by the customer. The business process perspective looks at each knowledge asset in terms of how quickly knowledge can be applied i.e., the cycle time, quality and productivity. Finally, the balance scorecard looks at knowledge assets from the organisational learning perspective. That is, is the knowledge unique to the company and does it encourage market innovation and continuous learning.

Portfolio management, using tools such as the Boston Consulting Group (BCG) matrix will help business analysts to identify where existing knowledge assets are being applied.[18] Clearly, if key knowledge assets are being applied to 'dogs' there is a requirement for change.

Having identified and analysed the company's knowledge assets and attributed some value to them it is then appropriate to initiate knowledge asset actions. These actions can be summarised as follows:

- Initiate the development of future knowledge assets.
- Secure those knowledge asset which are considered to be vulnerable.
- Dispose of those knowledge 'assets' which yield no benefit.

To initiate the development of future knowledge assets and secure existing knowledge assets the company may have to consider one or all of the following courses of action; start new research and development (R&D) programmes, improve the existing technology transfer process, consider transferring people within the company, develop an improved information technology (IT) based knowledge sharing architecture, enter into collaborative actions with partners, and possibly consider changes to staff recruitment.

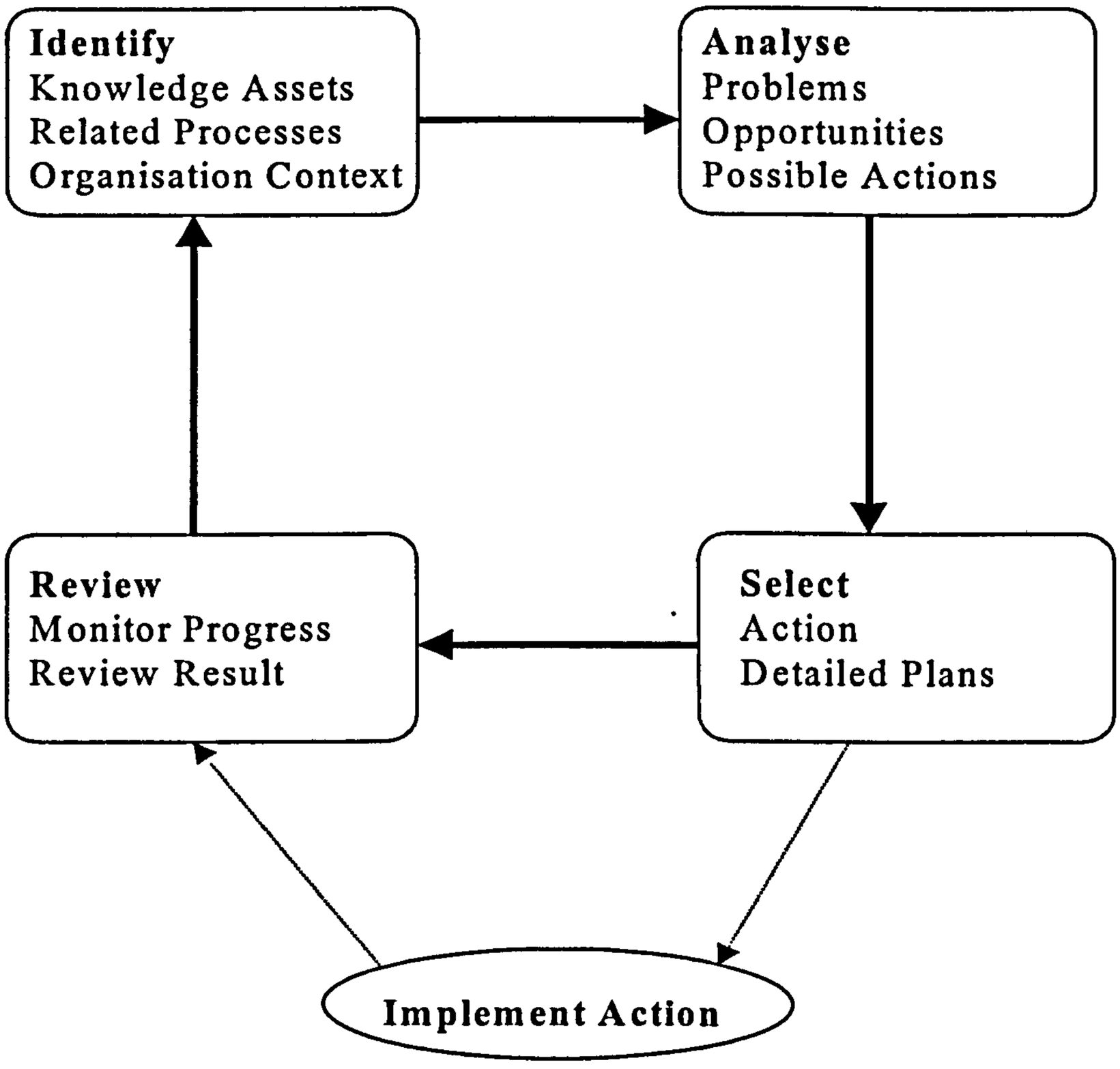



Figure 1.1, Knowledge Management Methodology

The corporate knowledge strategy must also embrace process management i.e., the management of those process that provide for the generation of knowledge assets. It is first necessary to identify the company's knowledge related processes. Typical knowledge related processes are:

- Acquisition processes
- Development processes
- Retention processes
- Transfer processes

- Dissemination processes
- Maintenance processes
- Validation processes
- In use processes
- Re-use processes

Part of the identification of knowledge related processes will involve the studying of the organisation's documentation with respect to quality, processes and procedures. This documentation should provide an insight into how acknowledged knowledge related processes should operate. In addition, it will be necessary to perform interviews with key members of staff to establish whether documented processes are known and followed. Interviews should also establish whether undocumented processes exist.

Having identified the existence of knowledge related processes it is necessary to look for evidence that they are being; performed, defined, managed and improved. It is also necessary that the effectiveness of these processes be measured. This is most effectively achieved through benchmarking studies.

As a result of analysing the knowledge related processes it may be desirable to introduce change to specific areas; new perspectives may be added to existing practices to encourage an awareness of knowledge assets; augment existing processes and practices with new tasks and define new roles, responsibilities and processes. This may requirement to introduce new or modified functions and structures. Possibly the most significant knowledge related process changes could be in the introduction of cultural incentives to encourage:

- The creation of new knowledge.
- The sharing of knowledge through improved networking.
- The making of knowledge re-usable such that all interested parties can use it.
- The maintenance and development of existing knowledge; ensuring that there is ownership and responsibility for knowledge
- The searching for and the re-using of knowledge; avoiding the 'not invented here' syndrome.

It is important that an IT based knowledge sharing architecture be introduced. Its primary purpose being to support knowledge asset related processes. The architecture should be a distributed system supporting:

- Off-line knowledge (pointers to people, external documents etc.)
- Normal text
- Structured text (possibly source code where appropriate)
- Explicitly represented knowledge

In addition the IT based knowledge sharing architecture should provide facilities for:

- An overview (map/index) of knowledge assets
- Searching for knowledge
- Recording and sharing knowledge
- Instrument processes to record the frequency of use and updates of knowledge assets, also the quality of knowledge

The principle operation of the knowledge management methodology alluded to above is illustrated in Figure 1.1. To summarise, it is important that any large scale changes that are introduced for managing knowledge must be well lead by senior management and be supported by all staff i.e., there must people commitment. It is necessary to model and benchmark existing processes and where improvements are desirable these must be prioritised. Where possible new practices should be introduced that build on existing ones. Consideration should be given to whether it is appropriate to introduce small or large scale changes. Finally, all changes that are introduced must be monitored in order to determine whether the desired results are attained.

The following section now discusses BAe MA&A's strategic and operational view of knowledge management. In addition, the section discusses the knowledge management issues that currently confront BAe MA&A and their response to them.

## 1.3. British Aerospace, Military Aircraft and Aerostructures Strategic and Operational View of Knowledge Management

At the time of writing, BAe MA&A does not possess a homogenous corporate knowledge strategy aimed at directing the process of knowledge capture and its subsequent utilisation. However, senior management at BAe MA&A are keenly aware of the importance of being able to manage knowledge in the company. Key knowledge management issues which confront BAe are discussed below.

The global market pressures on BAe MA&A to become leaner and more competitive have over the years lead to significant down-sizing and rationalisation. While this has lead to the compression of management hierarchies it has also put many of the traditional company structures for retaining product knowledge under pressure e.g., there has been a reduction in the number of apprenticeships. It is acknowledged that the effect of these external pressures are difficult to measure in terms of knowledge retention, precise metrics are difficult to define. While the loss of knowledge due to rationalisation and normal staff turnover has never directly affected the 'bottom line' there have been instances where staff loss has had an adverse impact. There have been in the past occasions where there have not been enough skilled people for projects. As such, it has been necessary to buy in resources where skills were lacking or were there was an insufficient supply of skilled people to meet project requirements. Here one is in effect buying in knowledge. There have been instances of BAe MA&A employees who have been made redundant returning to work for the company in a consultancy role. Clearly, whilst at the time this may have been necessary in order to meet the skill shortage for a particular project, from a range of view-points such a situation is undesirable.

It is important to note that the effect of globalisation and rationalisation are not the sole pressures acting on BAe MA&A's knowledge retention capability. BAe MA&A is interested in getting stability into the workforce. It is not a viable proposition for BAe MA&A to try and match peak demand. If the company did so it would always be going through periods of hiring and firing and this would induce a feeling of insecurity into the work force; it would become progressively more difficult to attract the staff of the desired calibre.

It has been illustrated that the twin pressures of the global market and a requirement to keep stability in the workforce represent a significant influence on BAe MA&A's knowledge retention capability. In terms of providing the necessary knowledge resources for projects BAe MA&A matches available skills to current projects and then makes good any knowledge shortfall by recruiting staff or out-sourcing. This process is effective in terms of company knowledge retention providing there is a metric(s) in place which can effectively identify what represents an acceptable knowledge shortfall. Metrics which could be utilised to identify a knowledge retention shortfall include the following:

- The amount of rework that the company has to perform represents a measure of success with respect to retaining knowledge.
- The number of new contracts that the company is getting is an indication of customers belief that the company has the necessary knowledge in the right areas. It is worth noting here that contracts are won by a team and in the process of winning contracts it is necessary to present the appropriate technology to the customer.
- The ability of the company to attract staff of the desired calibre.

Whilst these metrics represent potential with respect to identifying knowledge at risk in terms of BAe MA&A identifying knowledge shortfalls, there is a requirement for further research to be performed to draw these metrics (and possibly others not mentioned here) into a homogenous entity. This exercise while desirable is out of scope of the study discussed in this thesis.

The loss of knowledge or capability from the company is only fully appreciated when it is required to re-apply this 'know-how' again. It is often the case that once a project is completed it is rightly assumed that BAe MA&A has expertise in a particular area. However, if care is not taken this assumed knowledge can be diluted over time. It is important to note that knowledge loss is not due solely to staff turnover, advances in technology also have a role to play with respect to knowledge retention. For a company such as BAe MA&A to retain its competitiveness in the military aircraft industry it must embrace new technology in a proactive manner. The fact that BAe MA&A is a leader in many areas of new technology development is one of the principle factors that facilitates its capability to attract high calibre staff. Naturally, all staff want to be involved with new technology i.e., it improves their skill base and makes them more marketable. While this situation has obvious benefits for BAe MA&A, care must be taken to ensure that the company does not find itself in a situation where it can do the futuristic thing but not the traditional alternative. As an example, it is unlikely that BAe could build the Experimental Aircraft Programme (EAP) as it was first conceived. If a new EAP were to be built, by the very nature of the advancement of technology the aircraft would have to by necessity embrace new technology. Clearly, BAe MA&A needs to balance its ability to absorb new technology with the requirement to retain basic skills in an environment where everybody wants to be involved with the futuristic programmes.

A further factor which makes knowledge retention difficult within BAe MA&A is the length of aircraft projects. An aircraft project may last 20 to 30 years. As such, if care is not taken, the knowledge and associated skills that were available at the

conception of one aircraft project may not be readily available with the start of the next. Clearly, keeping skills current over considerable periods of time is a problem. BAe MA&A place the responsibility for keeping skills current with the functional managers. It is their responsibility to identify skills gaps and initiate the action necessary to fill them. The principle method of keeping skills up-to-date is to instigate special R&D technology projects whose sole purpose is to build and consolidate skills. These projects are expensive and time consuming i.e., these projects do not contribute to the 'bottom line' in their own right. One of the issues that confronts management is how much R&D should people do to keep their existing skill base up-to-date and to learn new skills.

A further knowledge retention issue which it is appropriate to touch on here is the matter of out-sourcing work. It is acknowledged that there will be a tendency for BAe MA&A to out-source increasingly more work in the future. The concern here is that BAe MA&A must be certain that it only out-sources straight-forward mechanistic production or design work, and not work which could in the future provide the company with a competitive advantage. Whilst this is an obvious target for BAe MA&A to aim for, its achievement is difficult e.g., the need to meet project deadlines may place work which would otherwise be desirable to keep in the company in the hands of third party contractors.

The above discussion has highlighted many of the knowledge retention issues that face BAe MA&A and the aircraft industry in general. It is appreciated throughout the aircraft industry that computer based tools and AI technology in particular, have a significant role to play in facilitating the capture of product knowledge and thus alleviating some of the pressures on knowledge retention. The research project documented in this thesis represents part of BAe MA&A's research into the use of AI technology for the management of knowledge. In strategic terms this project proves the viability to BAe MA&A of employing AI computer based tools for capturing and delivering product knowledge to the points of need.

The issue of knowledge acquisition is very important to BAe MA&A in the context of the company retaining its competitive position in the military aircraft market. For BAe MA&A the principle knowledge management issues to be addressed in area of knowledge acquisition are:

- Delivering newly acquired knowledge to the points of need
- Strategic partnerships
- Monitoring knowledge external to the company

Technology acquisition is an important issue with respect to the management of knowledge. New technology can be developed in-house or the work can be out-sourced to a third party. However this new technology is developed there is a requirement to deliver the technology to the points of need. If technology development is in-house there is a phase in the technology development process that delivers this technology to the project in question. This is usually a straight forward process as new technology can be directed to projects by transferring the appropriate staff with the necessary skills to the project team. In the case of out-sourcing new technology development, the process of delivering the technology to projects is rather more complicated. Here there is an additional stage in the technology transfer process; it is

first necessary to get the technology in-house from where it was developed. The transferring of technology can be a significant issue. If the technology is software based then there is no great technology transfer problem. However, if the out-sourced technology developed is a skill or expertise embedded in a university or comparable institution then the transfer of the technology is not a simple task.

Because BAe MA&A have had intractable problems with respect to technology transfer in the past a new initiative was introduced. This initiative requires that a feasibility study be performed on any project that will require some form of technology transfer. The project will only be allowed to continue if it can be shown to be feasible to transfer the new technology into the company. The new technology is studied in detail to determine what the technology 'looks like'. It is possible that the new technology may require further work to be performed on it so that it can be used within the company. With respect to the research project documented in this thesis, it is software based and fully documented, thus the useful information it possesses while not being utilised directly by a particular project is readily available to BAe MA&A staff as a source of reference.

In some instances it is not always practical to attempt to generate significant quantities of new knowledge in-house. This may be due to time constraints and/or the cost of the R&D required. Equally, in such situations it is may also be impractical to attempt to pay for this knowledge to be developed by a third party. In such a situation BAe MA&A may consider company acquisitions or entering into strategic partnerships in order to acquire the desired knowledge.

To date, BAe MA&A has generally preferred to enter into strategic partnerships in order to gain additional knowledge. These strategic partnerships may relate to significant projects aimed at building knowledge and relationships in the long term e.g., the Joint Advanced Strike Technology (JAST) project, or be the means to provide a forum for the exchange of knowledge in the relatively short term i.e., the focus is placed on a specific technology. For example, BAe has specific skills in the area of designing and manufacturing super plastic formed, diffusion bonded (SPF/DB) titanium structures, in particular with respect to x-core configurations. To obtain additional knowledge relating to cellular core SPF/DB titanium structure configurations it was deemed desirable to enter into a technology exchange agreement with Dassault Aviation. This technology exchange agreement provided BAe MA&A with some insight into the design and production of cellular core SPF/DB. Clearly, technology exchange agreements are a two way process and as such care must be taken to monitor the flow of knowledge in and out of the company.

BAe MA&A is aware of a large reserve of knowledge outside of the company e.g., residing in universities, R&D centres, within other companies and on the internet. A significant issue that confronts BAe MA&A is how to tap into this 'pool' of knowledge. BAe MA&A do a considerable amount of technology scanning. The technology scanning process is followed by an acquisition process. However, it is not possible to scan everything. It is relatively easy to direct resources for knowledge capture when you know what knowledge you wish to acquire. However, where it is uncertain or not known what knowledge will be required in the future it is harder to direct knowledge capturing activities in the right direction.

The problems presented through the desire to monitor the external knowledge reserve are not peculiar to BAe MA&A, all companies are confronted with the same issues. BAe MA&A's primary method of keeping abreast of advances in technology which may be of benefit to company in the future is to operate an informal benchmarking system which is embedded within the company's philosophy i.e., everyone is involved. Staff are encouraged to go on company visits, attend conferences and report back on new products, processes and methods as they come across them. Where it is discovered a particular company or institution has expertise in a certain area this is noted and logged. Clearly, this benchmarking process is a significant driver for BAe MA&A's future investment plans. This benchmarking process will shortly be enhanced through the introduction of the BAe MA&A virtual university. This will enable staff to not only gain access to new knowledge in a more rigorous manner but also provide a forum for the discussion of methods they've seen utilised elsewhere. Clearly, to continue to be a successful method of scanning the knowledge that resides outside the company staff must continue to be encouraged to participate proactively in this process.

As a forum for exchanging ideas the BAe MA&A intranet has considerable potential. At the time of writing the intranet within BAe MA&A is not interactive. However, there is a pilot project being run by R&D to assess the most appropriate method to deliver an interactive capability company wide.

To conclude this discussion of knowledge management at BAe MA&A; it appears that BAe MA&A prefers not to manage knowledge as a separate discipline but have knowledge management embedded within the existing separate functional disciplines. The problem here is that there is no strategic direction for the knowledge management that takes place at operational level i.e., operational knowledge management can only be co-ordinated company wide if there is a corporate knowledge management strategy to follow i.e., preferably documented where possible. By their very nature functional departments do not have the capability to take a global perspective.

It is important to point out that whilst no corporate knowledge strategy currently exists at BAe MA&A there is certainly an appreciation of the need to manage knowledge from a global perspective. The new design methodology which is presented in this thesis takes such a perspective; it looks at the entire design process irrespective of functional departments. The following section now provides an overview of this proposed new design methodology.

## 1.4. Overview of a Proposed New Design Methodology

As indicated at the beginning of this chapter, the retention of and subsequent re-use of product knowledge and the management of intellectual capital is an issue that confronts not only BAe MA&A but the military aircraft industry as a whole. The pressure to reduce costs and yet maintain innovative design is the rational behind the need to manage knowledge effectively.

Because traditional design methods do not foster knowledge retention and re-use there is a need for a new approach to the design process. This study, through the application of artificial intelligence (AI) technology presents a new design methodology whose underlying aim is to capture product knowledge and deliver it to the points of need in the design process. It should be borne in mind that the issues addressed in this study are not peculiar to BAe MA&A but are confronting every 'player' in the military aircraft industry.

This thesis introduces the new design methodology for BAe MA&A by first exploring the limitations of the company's current design methodology in the context of it being able to capture product knowledge and subsequently facilitate the utilisation of this knowledge. Having exposed the said limitations of the current design methodology these became the basis on which the new design methodology was developed. That is, the over coming of the limitations of the current design methodology was the starting point for developing the new methodology.

This thesis provides the reader with a broad overview of the new design methodology and then proceeds to focus on the conceptual design area of this methodology. This is where the detailed research throughout this study was primarily focused and where the contribution to knowledge was derived. Research in the conceptual design arena has lead to the development of the Intelligent Conceptual Engineering System (ICES). As will be apparent, the majority of this thesis is subsequently taken up with a technical description of ICES.

Having indicated that ICES represents the significant topic under discussion in this thesis it is appropriate here to provide the reader with a simplified 'picture' of ICES to relate to before leading onto any technical discussion. As such, the remainder of this section provides this picture.

A significant influence on a design aid, such as ICES, is the viewpoint of the design methodology in which the tool resides. For instance, the design methodology as viewed from the aerodynamic perspective is significantly different from that of manufacturing or the drawing office. It is essential to identify the appropriate viewpoint prior to developing the system further. In the context of this study the viewpoint taken was that from the Structures Unit.

ICES working in the conceptual design environment applies the AI disciplines of knowledge-based systems and a case-based reasoning (CBR) to the knowledge capture and retention problems presented in this arena. Looking first at the knowledge base. The knowledge-based system (KBS) applies generic rules to identify the 'best structure' in the context of the manufacturing and structures disciplines. Its operation is implemented from the top-level i.e., the system asks the user what the component in question that it is desired to design offers to the aircraft. On the basis of the inputs entered by the user, the knowledge base identifies the design drivers that act on the design process. With subsequent further interrogation of the user, the ICES knowledge base identifies the most appropriate or best structure. Naturally, having identified a preferred structure via the KBS the user will wish to see the design case in the case base that most closely conforms to the structure selected. This case is presented to the user.

Turning attention to the case base. The case base operates under the premise that the user has access to specifications of some description. The user enters these specifications and the case base through a range of techniques presents the best matching case to the user. As will be described in greater detail in chapter 6, aircraft cases are relatively few but can be enormous and as such traditional case base search methods are not satisfactory when used on there own. The case base methodology discussed in this thesis introduces the reader to case selection using a new highly novel technique called the 'jury technique'. Broadly speaking, this technique requires that the best case as presented by the system defend itself against alternative design case solutions.

All new theories have to have a mechanism by which they can be validated. This may be a one-step or evolutionary process. In the context of the ICES methodology this required the development and evaluation of a series of software prototypes. It is important that the reader appreciate that prototyping is the appropriate method to tackle this problem. The development of a new design methodology is largely an abstract exercise. There are few hard-and-fast rules with respect to what is acceptable and what is not. However, new ideas and concepts have to be transferred from the abstract into a level of reality. The prototype provides this element of reality and represents the test-bed for new ideas and concepts. Only when the methodology has been tested and fine-tuned in the safety of the prototyping environment can consideration be given to taking the concepts through to the next stage of development i.e., a partial or even full implementation.

Having provided this brief overview, Chapter 2 consists of a literature review. The chapter places the work documented in this thesis in the context of previous research performed in the area. In addition the chapter highlights the significant differences between this study and what has previously been researched.

# Chapter 2

# Literature Review

This Chapter is divided into three sections. The first section places the research documented in this thesis in the context of previous work in the area. From this discussion it will be apparent that knowledge-based system (KBS) and case-based reasoning (CBR) technology have had a prominent role to play within this study. The second section reviews prior successful applications of KBS (expert systems are included here as they are a sub-set of KBS) and CBR. The discussion focuses on the general application of these artificial intelligence (AI) tools and where they have been applied in the design environment. The final section of this chapter discusses how the application of knowledge-based systems and case-based reasoning as applied in this research differs significantly from previous work which has applied these technologies.

## 2.1. Placing the Research in Context of Previous Work

The concept of finding, mathematically or otherwise, the optimum structural configuration to satisfy a specific design problem has a long technical history. For the most part this has involved seeking the minimum weight of the structure subject to the satisfaction of constraints placed on behavioural responses of parameters such as stiffness, stress, aeroelastic phenomena etc. as described in reference 19. This approach has been very effective in improving designs but is mainly directed to the final phase of the structural design process and is concerned with structural performance only. The strongest driver in the development of these methods in the last quarter century has been the quest to design high performance military aircraft where weight reduction is of exceptional importance.

Although weight will always be a very important factor for the designer of military airframes it is now being recognised that other factors relating to cost, manufacturability etc. are equally or more important in defining the optimum structural design. In response, the last decade has seen a concerted attack on broadening the scope of structural optimisation to include the effect of a range of non-structural factors. An approach to this more general problem has been put forward by Sobieski[20] in which mathematical models of physical phenomena are coupled together. This has been employed in a number of applications involving the design of aircraft structures taking account of structural, aerodynamic and other factors.[21,22] This has great benefit when the phenomena influencing the design can be modelled mathematically and has been the subject of recent papers[23,24] which reflect developments taking place in a major European multi-disciplinary optimisation (MDO) project funded by the European Union.[25]

The research project presented in this thesis extends the optimisation process to cover the requirements of the total design process. This goes beyond the work reported in the above references and requires the incorporation of knowledge and information relating to manufacturing and process control. Such phenomena are fundamentally different from those considered earlier and cannot be easily represented

by simple mathematical models. A number of contributions have been made in this field. Choi et al[26] have brought together a set of tools to support the concurrent design of large scale tracked vehicles which involves a cost benefit analysis employing conventional optimisation as a component. Dobbs et al[27] have shown how concepts of affordability can be incorporated into the conventional airframe design process. Thompson[28] has indicated how some of the modern technological developments can be combined into an integrated airframe design system. Tyll et al[29] propose a design methodology which integrates aerodynamic shape and cost for high speed magnetic levitation vehicles (MAGLEV). The current version of the tool addresses operating cost, acquisition cost, and life cycle cost and performs a 2D side view aerodynamic analysis. The modular set-up of the methodology allows for its enhancement as new models are developed. Mistree et al[30] introduce a decision based design methodology called the decision support problem technique (DSPT). Within this methodology the concept of a 'satisficing' system design is presented with respect to constraints and multiple goals. The approach emphasises compromise with respect to design modifications by making appropriate trade-offs based on criteria relevant to the feasibility and performance of the system. A satisficing solution being that point which achieves the system goals as far as possible. Vadde et al[31] introduce a method for accounting for uncertainty in the environment or technology at different stages in the design timeline by incorporating the mathematics of fuzzy set theory into decision support problems, specifically the compromise decision support problem. The concept introduced is where a decision-based design process is modelled as the design evolves, that is, as the design problem becomes more precise. Schrage et al[32] present a methodology for aircraft producibility assessment which utilises a KBS for manufacturing process selection, that addresses both procedural and heuristic aspects of designing and manufacturing of a high speed civil transport wing. Hale et al[33] introduce a method of implementing integrated product and process development (IPPD) through a decision-based formalism. The approach has two parts. Firstly, an architecture called DREAMS facilitates design from a decision making perspective. Secondly, DREAMS is supported by a computing infrastructure called IMAGE.

Prior work in the field readily acknowledges the need for applying formalisms and methodologies to the design process. However, much of the prior work does not address 'real' world problems. Researchers have fallen, inadvertently, into the trap of finding a solution to an abstract problem and then attempting to apply this to a real world problem. The result being that, at best, the solution can only be applied to a limited real world problem domain. In addition, it is often the case that the solution proposed fails to encapsulate the real world problem under consideration.

Without a real problem to solve at the outset of a study it is relatively easy to talk about design spaces and satisficing solutions as done by Mistree et al. These types of solutions mean little to the designer in real terms. The designer wants real solutions to his or her problems i.e., explicit rather than implicit guidance. This study differs significantly from prior work in that it addresses a real problem and provides a solution that can be readily comprehended. That is, the optimising component of this study identifies real structural types and manufacturing scenarios; the problems that the requirements of these components of the design process present are dealt with in a realistic manner that is both comprehensible and useful to the designer.

Work in the area aircraft optimum design has to date has largely failed to address the utilisation of past designs to assist with the solving of current design problems. This is because aircraft design information is unwieldy and difficult to put in a format which facilitates its ready re-use in the design environment. The present study utilises concepts associated with modern artificial intelligence (AI) applications and places them in the design process. Specifically, knowledge-based system and case-based reasoning are employed to facilitate the capture and effective re-use of prior design cases. That is, the KBS identifies a 'best structure' from a manufacturing and structures perspective and armed with this knowledge draws previous design cases from a case base which can be utilised in the design process. The approach taken to derive a best structure differs from previous work in the area. The work presented in the current study, acknowledges that it may not always be possible to derive a satisfactory solution. A novel solution to the problem has been identified which allows the application of secondary rules residing in the KBS, which permits the designer to view the problem from different perspectives (manufacturing and structures in this study) and thus, view the solution from these perspectives.

While the approach outlined in this study focuses on applying the AI technologies of knowledge-based systems and case-based reasoning, it is appropriate at this juncture to acknowledge that an AI-based approach to design decision support is not the only approach available. The following paragraphs now discuss the approach offered by Multiple Attribute Decision Making (MADM).

The approach presented by MADM represents a mathematical-based approach to design decision making. The rigorous nature of MADM encourages numerical values to be assigned to objectives, attributes and consequences. The paradigm of decision analysis used in MADM is based around a five-step process[34]:.

- Pre-analysis
- Structural analysis
- Uncertainty analysis
- Utility or value analysis
- Optimisation analysis

Whilst it would not be appropriate to discuss the discipline of MADM in detail in this thesis, it is proposed to give an overview of the above five-step approach and indicate why aspects of this approach were considered to be unsuitable for the study documented in this thesis.

At the pre-analysis stage the problem and all its viable solutions are identified. Decision analysts admit that an insightful generation of alternatives is of paramount importance. They also note the often overlooked fact that good analysis of a set of existing alternatives may be suggestive of ways to augment the set of solutions.

The pre-analysis approach assumes that the person(s) performing the pre-analysis has a good knowledge of the domain area. Clearly, in a complicated technical knowledge domain the analyst would have to possess a significant level of expertise. As will be apparent later in this thesis when ICES is discussed, the decision support

tool developed does not have a pre-analysis stage. The reason for this lack of pre-analysis phase relates to a significant difference between MADM and the design decision support tool presented in this thesis. MADM tends not to support a design development environment where solutions to problems are developed but rather an environment where through a process of numerical manipulation choices are made between known solutions.

The structural analysis stage of the decision analysis process employed by MADM requires the designer to structure the qualitative aspects of the problem. That is, what choices should be made now; what choices can be deferred to later in the design process; and what design choices are not under the designers control. The information available for the structural analysis process may be structured in an orderly manner e.g., using a decision tree for example.Clearly, structuring of the decision process is a relatively straight forward task where the design problem is easily dissected and the impact of design drivers and constraints is explicit. However, where there are many design drivers (e.g. fourteen in this study) which impact on one another in a variety of ways depending on the view point taken then the rigorous structural analysis process may not be a viable or realistic option. Even with a software package to display a decision tree or equivalent structure it is envisaged that the analysis process would soon become confusing.

The KBS approach adopted in this study does not attempt to perform an up-front analysis of the structure of the design problem. The KBS rather structures the anatomy of the design problem in reverse. That is, the KBS interacts with the user, fires rules and arrives at a solution. Then through the use of an audit trail, the user is able to explore the anatomy of the solution. An effective audit trail will quickly draw the users attention to important decision areas of the derived solution.

The uncertainty analysis stage of the decision analysis process employed in MADM requires the analyst to assign probabilities to the structure of the design problem defined in the structural analysis stage. The assigning of probabilities will be supported by empirical data, dynamic models and expert testimony. The ease with which probabilities can be assigned will depend largely on the availability of supporting information. As with structural analysis, uncertainty analysis is a relative straight forward task where the design problem is easily described. Conversely, uncertainty analysis becomes considerably more difficult where the design problem is large and difficult to decompose. As the KBS approach employed in this study is heuristic by nature uncertainty analysis is not the issue it is in MADM.

With utility or value analysis, the designer assigns utility values to consequences associated with the various options or paths presented by the decision tree. In any design problem there will be associated costs and benefits which may be attributed to any specific path or course of action. Assuming that the designer is able to assign numerical values to each of the courses of action or paths in a decision problem he or she is able to rank relative preferences. As with uncertainty analysis, this process assumes the designer is able to assign justifiable numerical values to the consequences associated with the various courses of action attributed to any particular design. It is considered that value analysis is relatively easy to perform at a high level of design

abstraction but becomes more difficult to perform at the detailed design stage. In addition, where there are many courses of action, determining the precise consequences associated with any particular course of action may not be practical. Where the consequences of a particular course of action are reliant on tacit knowledge the performing of an accurate value analysis presents difficulties. That is, there needs to be a mechanism to draw out this tacit knowledge before values can be assigned to it.

After the designer has structured the design problem and assigned probabilities and assigned utilities, the optimal strategy may then be determined. The optimal strategy will be the solution that maximises expected utility. There are various techniques that may be employed to obtain the optimal strategy. The simplest is the dynamic programming algorithm of averaging-out-and-folding-back.

It should be appreciated that the paradigm of decision analysis utilised in MADM is a rigorous mathematical approach and is similar to traditional structural optimisation in this respect and it too is of great benefit when the phenomena influencing the design can be modelled mathematically. Both approaches to be successful for a design problem must facilitate the assigning of numerical values to the component parts of the problem to be solved. Before closing this discussion with respect to MADM it is proposed to summarise the issues raised during the study of this particular paradigm.

MADM tries to make designers think about the decision making process. The rigorous nature of the discipline encourages numerical values to be assigned to objectives, attributes and consequences. The approach is highly successful where the designer is in a position to apply numerical values to the design decision making process. Clearly, this is not always an easy or practical option. In this context, the approach suffers from the same limitations as traditional structural optimisation i.e., as alluded to ealier in this chapter.

In MADM bounding the problem is critically important, otherwise there is a danger of arriving at a sub-optimal solution. In addition, it is very difficult to rank consequences as MADM requires in the aircraft design environment. This is due to the duration of an aircraft design i.e., typically 20 to 30 years. There is a need to predict the future if consequences are to be ranked accurately. For the same reason value analysis results may be difficult to apply in an aircraft design environment as the benefits further down the time-line may be difficult to appriase.

MADM is reliant on there being access to explicit knowledge of the design problem in question. Tacit knowledge is not easily encapsulated within MADM. This does not ignore MADM's ability to support trade-offs with respect to subjective view-points.

One of the principle reasons for first considering applying AI technology to the aircraft design environment was that formulated correctly, knowledge based and expert systems are able to extend expertise throughout an organisation. The decision support methodology embedded within the ICES software prototype may readily be

employed BAe MA&A. By its very nature MADM does not address the issue of extending decision making expertise in the knowledge domain to which it is being applied.

An important issue that makes MADM not appropriate for this study is that MADM is effective for selecting between design preferences but not for actually arriving at the design preferences in the first place. Thus, in order to be effective MADM assumes a degree of domain knowledge. As indicated above, as knowledge based and expert system technology facilitates the extension of expertise throughout an organisation there is no requirement for the designer or user to have significant domain knowledge.

The work presented in this thesis differs in another crucial respect with previous work. There is the underlying intention throughout the new methodology presented to capture product knowledge. Whilst other work does facilitate the capture of some degree of product knowledge e.g., Schrage et al, there is not perceived to be this underlying intention. The methodology presented in this study addresses the capture of product knowledge at all stages in the design process i.e. conceptual, preliminary and detailed design.

The approach presented blends with the analytic structural optimisation methods and provides the design engineer with a pathway for incorporating other important design drivers such as manufacturing, labour, inspection, assembly, and start-up costs into the preliminary design phase. These are normally deferred to later in the design process. In addition, the new design methodology presented in this thesis acknowledges the very important role that research and development (R&D), and company business drivers have to make in the design process. While acknowledging the work by Allwright,[35] much of the prior work in the field has conspicuously failed to reflect the important impact that these areas can and do make in the design process.

To summarise, previous work has been extremely good for dealing with mathematically based problems e.g., MADM and traditional structural optimisation. There has been an appreciation that there is a need to embrace broader aspects of the design process i.e., beyond the traditional scope of structural optimisation. With this in mind, recent research as indicated above, has expanded the focus of the design optimisation problem. However, the review of previous work has revealed weaknesses. Much of the work does not address real problems. The possibility of using past aircraft designs has largely been ignored. There has been little explicit intent to capture and re-use product knowledge. The role of R&D and company business drivers has again been largely ignored. It is these areas of weakness with respect to previous work that this study addresses.

## 2.2. Previous Applications of Knowledge-Based Systems and Case-Based Reasoning

### 2.2.1. General Examples of Successful Applications of Knowledge-Based Systems and Case-Based Reasoning

This section illustrates the wide range of knowledge domains that KBS and CBR technology has been applied to since artificial intelligence has become a legitimate subject of research.

DENDRAL was designed to infer the structure of organic molecules from their chemical formulas from mass spectrographic information about chemical bonds present in the molecules. Because organic molecules tend to be very large, the number of possible structures for these molecules tends to be huge. This problem of a large search space is addressed by applying the heuristic knowledge of expert chemists to the structure elucidation problem. It represents one of the earliest expert systems and was developed at Stanford in the late 1960s.[36]

The research at Stanford also gave rise, in the mid-1970s, to MYCIN which was one of the first programs to address the problem of reasoning with uncertain or incomplete information. This expert system uses expert medical knowledge to diagnose and prescribe treatment for spinal meningitis and bacterial infections of the blood. Written in INTERLISP, a dialect of LISP, with around 450 rules within its knowledge base, MYCIN established the methodology of contemporary expert system implementation.[37]

XCON is an expert system for configuring DEC computers. In 1981, XCON had about 500 rules and could configure the VAX 780. It was progressively refined until in 1984 with about 4000 rules, it configured most of the DEC product line.[38]

PROSPECTOR was developed at Stanford in the late 1970s. It is a diagnostic expert system for determining the probable location and type of ore deposits based on geological information about a site. Its knowledge base contains about 1600 rules. Bayesian probability is used for treating uncertainties in information and rules.

INTERNIST is an expert system for performing diagnosis in the area of internal medicine. Developed further (with the name CADUCEUS), its knowledge base includes about 500 diseases, 350 disease manifestations, and about 100,000 symptomatic associations. CADUCEUS covers about 25 percent of the diseases of internal medicine.

CASEY combines CBR and causal reasoning to provide a causal explanation of a patient's heart disease symptoms. It uses case-based reasoning to recall and remember problems that it has seen before, and it uses a causal model of its domain to justify reusing previous solutions, to guide their adaptation for a new situation, and to solve unfamiliar problems.[39]

CABARET is a domain independent shell that integrates rule-based and case-based reasoning to facilitate applying rules containing ill-defined terms. The integration of these two reasoning paradigms is performed via a collection of control heuristics that suggest how to interleave case-based methods and rule-based methods to construct an argument to support a particular interpretation.[40]

CLAVIER generates autoclave load schedules by working interactively with a user. It presents a graphic depiction of recommended layouts, along with a prioritised list of parts waiting to be cured. The user enters or adjusts part priorities and selects from recommended layouts. The user can modify recommended layouts, in which case CLAVIER critiques the new (adapted) layout by comparing it with other layouts in the case library. Clavier uses cases in three ways: as constraints in its heuristic planning and scheduling module, to determine allowable layouts within the autoclave, and to help validate manually adapted cases. CLAVIER has been in continuous use at Lockheed since 1990. It began with 20 cases and now has over 300.[41]

JULIA does design in the domain of meal planning. It uses cases to propose plausible solutions, decomposing the problem as necessary and posting constraints to guide synthesis. JULIA exploits a repertoire of adaptation methods to transform previous meals and dishes in order to meet constraints on the current problem. These adaptation methods are used to both modify previous cases and to repair previous decisions that have been invalidated by constraints that arrive late.[42]

### 2.2.2. Applications of Knowledge-Based System and Case-Based Reasoning in the Design Environment

This section provides examples of where KBS and CBR technology has been applied in the design environment

A widely cited design expert system is HI-RISE developed by Maher.[43] HI-RISE is an expert system for the preliminary structural design of high rise buildings of rectangular shape. The major concern of HI-RISE is to generate feasible configurations, to the level of detail needed to make a selection from amongst alternatives, and to provide the initial estimate of geometric and mechanical properties for a detailed structural analysis. HI-RISE represents the design knowledge in the form of schemas and rules. The schemas contain the description of the design sub-systems and components, and the rules represent design strategy and heuristic constraints.

An expert system was developed to assist in finite element analysis (FEA) to analyse deformations and stresses in physical structures. In order to design a numerical model of a physical structure, it is necessary to decide the appropriate resolution for modelling each component part. Choosing the appropriate resolution, or finite element mesh, requires considerable expertise. The inductive logic programming algorithm called Golem was used to form PROLOG rules for this mesh generation expert system.[44]

SACON eases structural analysis by suggesting specific analysis strategies.[45] The system uses knowledge about stresses and defections of the structure under different load conditions to determine the appropriate strategy. SACON is designed to identify the type of numerical analysis, the required modelling detail, and the specific analysis data required. Decisions are constructed on the basis of material behaviour, relations between geometry and structural behaviour, measures of the importance of time and temperature changes, and user-supplied specifics such as characteristics of the

spectrum of analysis types, the relation between accuracy and model detail on the structure, its mechanical loadings and its temperature states. The system is rule based, employs backward chaining and was implemented using EMYCIN which is derived from MYCIN.[46] The system was developed at Stanford University.

A KBS was developed to assist in the manufacture of distribution transformers. It provides a production price estimation and design guidelines. The system is able to provide price quotes and bids, and details on the transformer design. It also generates parts lists automatically. It uses several knowledge bases that communicate over a blackboard. It also employs rules, frames, and neural networks. The system was implemented using PROLOG and was developed on a PC at Sistemas Inteligentes, Monterrey, Mexico.[47]

Adelli and Balasubramanyam developed an expert system prototype for optimum (weight) design of bridge trusses subject to moving loads.[48] This prototype (called BTEXPERT) has a the knowledge base which consists of the domain specific knowledge and control knowledge. The domain specific knowledge consists of parameters and rules where rules are in the IF THEN form. The control knowledge consists of control commands for solving the problem. The inferencing has both forward and backward chaining but does not address structures other than the truss structure.

Harris et al developed an expert system package as an interface to the STARS optimisation package.[49] This system addresses problems of algorithm selection, quality of convergence, and constraint satisfaction. The expert system was written using FLEX (FORTRAN Library for expert system) developed by RAE. The knowledge used in FLEX comprises rules and facts. Rules are presented in standard production rules. FLEX supports both forward and backward chaining inferencing. This expert system does not have the capability to learn from experience.

Berke, Patnaik, and Murthy investigated the application of artificial neural networks to capture structural design expertise.[50] An artificial neural network code, NETS, was used. A set of optimum design data were processed to obtain input and output pairs, which were used to develop a trained artificial neural network with the code NETS. Optimum designs for new design conditions were predicted by using the trained network. The nature of the neural network in producing the solution makes it difficult for users to understand the reason of any decision given by the network. This makes it unsuitable as an optimisation training media for non-experts.

ARCHIE is an interactive design-aiding system for architectural design, built using REMIND. It supports construction and evaluation of solutions. Users specify their problem description and/or solution description; the system retrieves and displays past designs and provides suggestions and warnings. In support of evaluation, the system computes potential outcomes and retrieves and displays past designs with similar outcomes.[51] This case base illustrated that cases can be very large and need to be decomposed into smaller units. In addition this case base showed that libraries of design cases can be very useful but may need to be supplemented with other types of design knowledge.

BOGART helps a user to design circuits. The user describes his or her new situation to the system, retrieves a design plan from the library, and selects a portion to replay. BOGART decides which decisions are applicable to the new situation and replays them. The user makes further decisions that are need to complete the design.[52] This case base showed that the applicability of derivational replay in design seems to be dependent on the applicability of the top-down refinement model of design to the design problem. BOGGART was applied to a mechanical design task, but it did not help much, because the task was ill suited to the top-down refinement model, it was really a parameter design problem.

CADET is a case-based design tool that uses index elaboration (situation assessment) as a means for finding and synthesising entire cases or parts of cases. Its indexing is in terms of abstract behaviours. New devices are created by synthesising the behaviours of pieces of known devices. CADET's abstract indexing and situation assessment procedures allow it to generate innovative design alternatives and to retrieve and synthesise device parts across domains.[53] This case base showed that the larger the supporting database, the better.

CADSYN, a hybrid CBR design model, integrates case-based reasoning with problem solving decomposition and constraint processes. Solutions are derived by finding the most relevant previous design situation and transforming the potential solution to fit the new design situation using a domain specific constraint satisfaction approach. Design cases were acquired from drawings of building projects designed by a local engineering consulting company.[54] This case base showed that design data cannot be collected from drawings alone. Drawings need to be interpreted by engineers involved in a project. Drawings by themselves do not make design criteria explicit.

KRITIK uses model-guided repair to synthesise new designs from old ones. Model knowledge is good for identifying needed fixes in a design and verifying that a design works, but it is inefficient for design synthesis. Cases are used in KRITIK for design synthesis, and adaptation procedures use model knowledge to identify needed fixes in a preliminary design, to suggest fixes, and to choose indexes for cases. Each design in KRITIK has a model of its behaviour associated with it. KRITIK creates models for newly-created designs by adapting old models. KRITIK has been employed on knowledge domains associated with physical devices e.g., simple electrical circuits and heat exchangers.[55] This case base showed that CBR is an efficient and effective method for solving simple design problems when case-specific models are available to guide the adaptation process.

Case-based design systems that have been developed using multimedia case representations include ASKJEF,[56] a case-based assistant for user interface design; ARCHIE-II,[57] a case-based aiding tool for architectural designers, and CASECAD,[58] a case-based design assistant in the domain of structural design of buildings. This approach is very popular and it is reasonable to suppose that many more multimedia case-based reasoning systems will be developed in the future.

To summarise, the strengths and weaknesses of KBS covered in this review fall in line with the well documented strengths and weaknesses of KBS in general. That is, they are very good where knowledge can be broken down into a series of rules. However, they are less effective when knowledge does not lend itself to this format. In addition, rules tend to be less 'secure' and prevalent at the boundaries of the knowledge domain. The case-based reasoners reviewed here appear to be good at handling a large number of similarly styled compact cases. The domains to which CBR has been applied to date facilitate validated cases being assembled relatively quickly. A weaknesses of CBR is that the discipline does not address the problem of what to do when cases are scarce. In addition, CBR appears to be unsuited for handling very large sprawling cases which take a long time to assemble and where knowledge may be incomplete. In the area of case adaptation, the knowledge domains to which CBR has been applied support inexpensive case validation. The problem of when the validation of adapted cases is very costly or even impractical has not been addressed. This study addresses the limitations of CBR as described here and presents solutions to these difficult problems.

## 2.3. The Application of Knowledge-based System and Case-based Reasoning Technology in a New Manner

The way KBS and CBR have been applied in this study differs significantly from how the technologies have been applied in the past. The KBS incorporates numerically justified design drivers into the decision making process. The method of assigning numerical weightings to the design drivers acting on the system is unique to this study and marks a significant contribution to knowledge. In addition, the application of secondary rules within a KBS as discussed above has not been encountered previously and considered to be one of the novel aspects of this work. The concept behind the rules operating in the KBS is novel, in as much that there is a requirement for a correlation between the rule requirement and the design drivers acting on the design process.

The case-based reasoning component of this study differs significantly from other case bases used in the design environment. The case base presented in this study operates a novel jury technique. The underlying concept is that it is a requirement of the system that the case presented to the user as being the 'best case' defend itself with respect to the next best case. The user(s) are the jury and decide whether the case presented as being the 'best' is indeed the best. This concept is a further unique contribution to knowledge and adds a significant new concept to the discipline of case-based reasoning. The requirement for the jury technique is explained in detail in Chapter 6 whilst, the operation of this new technique is documented in detail in Chapter 7.

Having reviewed existing work and placed the research documented in this thesis in context, Chapter 3 now proceeds by introducing the current BAe MA&A design methodology and its short-comings in terms of capturing and manipulating product knowledge.

# Chapter 3

# The Total Design Process

This chapter introduces the reader to BAe MA&A's current design methodology. As indicated in Chapter 1, the design methodology differs according to the viewpoint taken and it is essential to identify the appropriate viewpoint before progressing further. At the risk of repetition, in the context of this study the viewpoint taken was from the Structures Unit i.e., the structures perspective.

As indicated in section 1.4 of Chapter 1, this thesis presents a new design methodology for BAe MA&A based on the limitations of the current design methodology, which are alluded to in this chapter. However, it should be appreciated that in the context of managing knowledge, the development of the new methodology and subsequently ICES which are described in Chapters 5 and 7 respectively, a three phased process is applied. Firstly, from a strategic perspective it is necessary to identify the types of knowledge that reside in the current design methodology and how it is desired that this knowledge should be channelled. Secondly, it is necessary to identify the most appropriate processes and/or mechanisms to facilitate the capture and subsequent re-use of this knowledge. Finally, it is required to explore in detail how these mechanisms should operate. This chapter tackles the first of these three phases i.e., it identifies the types of knowledge present in the current design methodology and how it is limited with respect to being channelled or directed in the appropriate areas to facilitate the capture and subsequent re-use of product knowledge.

It is important at this juncture to emphasise that the very development of the new design methodology and subsequently ICES, is a knowledge management process in its own right. This starts at the top-level, where a strategic view of the current design methodology is taken in the context of it being a methodology for processing knowledge. This filters down to the lowest level of abstraction which relates to the implementation of the individual elements which constitute the knowledge management process as defined from the strategic perspective.

The discussion in the following section leads the reader in a logical step-by-step manner through the current design methodology. At each step of the process the type of knowledge generated and/or required by the methodology is indicated. In particular, the current methodology's limitations with respect to capturing product knowledge are highlighted and attention is paid to those areas where engineering resources are wasted.

## 3.1. The Current Design Methodology

Figure 3.1 illustrates BAe MA&A's current design methodology as viewed from the Structures Unit. The methodology naturally embraces the total design process i.e., taking on board preliminary design, conceptual and detailed design.

As can be seen from Figure 3.1, the current design methodology starts with the definition of the real structure. The real structure is comprised of three principle components:

- Non-structural mass distribution - this includes such items as hydraulics, wiring looms, paint and other protective coatings.
- Aerodynamics and aeroelastic constraint definition - this encompasses all aerodynamic and performance specifications.
- Design constraint definition - this encompasses materials, cost, manufacturing requirements and limitations, and possibly detailed customer requirements.

Whilst acknowledging each of these components are indispensable to the design process it should be pointed out that the aerodynamic and aeroelastic constraint definition will usually play the lead role. This is because the initial specification for an aircraft will stipulate certain performance criteria. These criteria will have a direct impact on the design constraints (e.g. material and manufacturing requirements) and the non structural mass distribution (e.g. the types of protective coating required).

To define the real structure there is a requirement for considerable interaction between these three areas. The process relies heavily on the skill and expertise of engineering staff. However, many of the decisions that are made are based not only on engineering skill but also on years of accumulated experience. In terms of knowledge management, these three areas represent knowledge assets which must interact with one another to achieve synergy to efficiently arrive at the real structure. However, currently there would appear to be a knowledge bottleneck in this area. Whilst there is access to an Engineering Data Management System (EDMS), which assists the process of deriving the real structure, an analysis of this area reveals that there is no mechanism whose purpose is to channel information from these three areas into one system and facilitate the capture and subsequent re-use of this combined experience. With no decision support system capability available to offer advice, the current system relies wholly on the expertise and experience of the engineering staff, however trivial the problem may be. That is, many design processes are repetitive and do not require the application of creative thought. An important point to note is that this process as it exists, even allowing for the full implementation of the concurrent engineering methodology, is wasteful of engineering resources and slows down the design process.

The limitations of the current design methodology, as a knowledge management process for capturing and storing product knowledge, has an impact with respect to the life cycle of aircraft projects. The length of an aircraft life cycle may be anything from 20 to 30 years, possibly even more. As such, the expertise that was present during the early design stages of any particular aircraft type may be in danger of being diluted over a period of time. Drawing the necessary expertise together again so it may again be utilised presents a potentially difficult task. While acknowledging the existence of company special projects aimed at keeping skills current, the duration of aircraft projects underlines the requirement for a focal mechanism, whose purpose is to channel the information from the above mentioned three areas into one system. Thus, provide a repository for aircraft design knowledge which is able to assist the current design problem being driven forward. As will be discussed in chapters 4 and

6 the introduction of AI technology in this key area facilitates the provision of this important focal point.

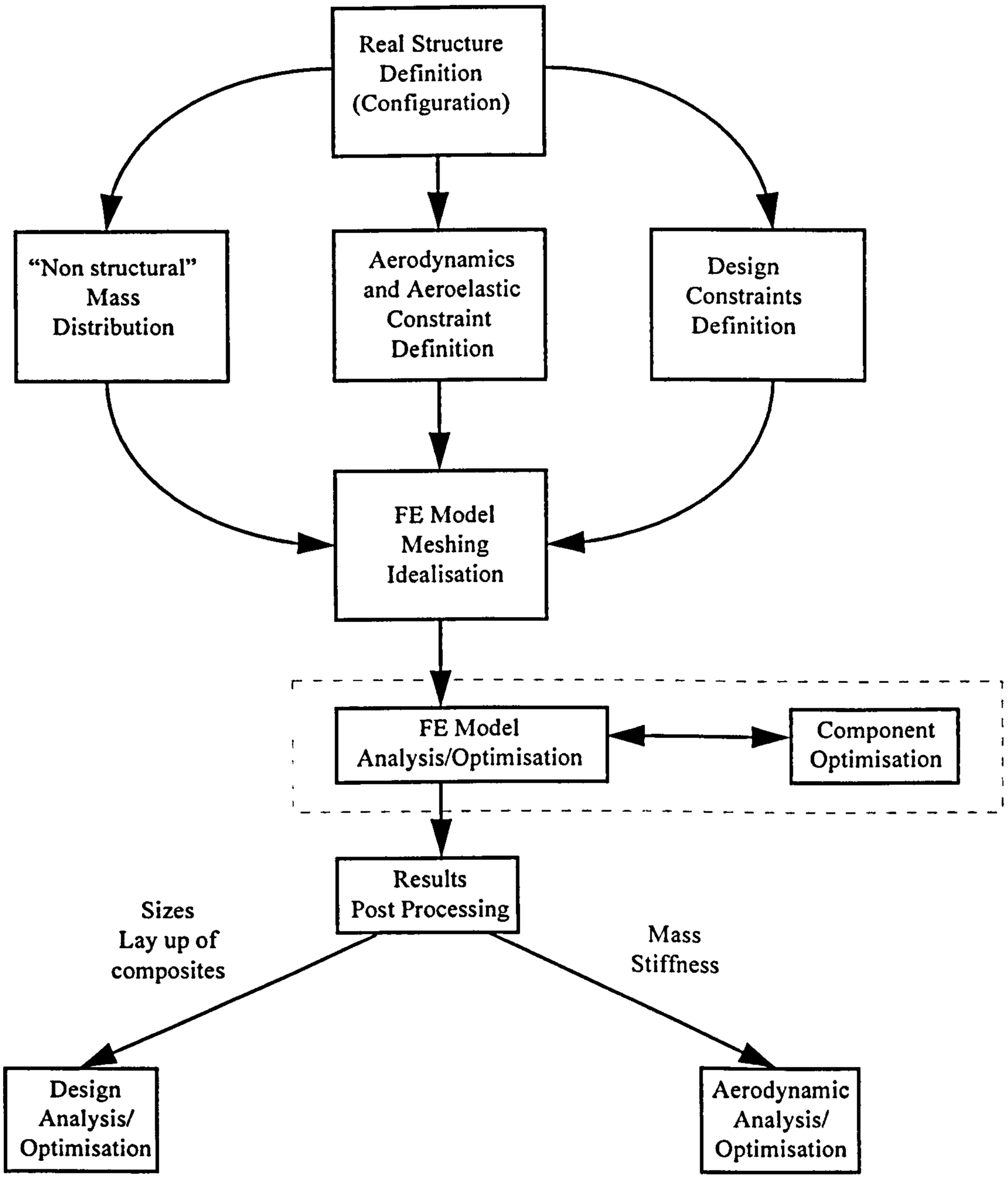



Figure 3.1, British Aerospace MA&A's Current Design Methodology
(as viewed from the Structures Unit)

Once the initial real structure has been defined it is then possible to generate the geometry within the CATIA computer aided design (CAD) package or possibly via a ground up approach through the finite element analysis package's pre-processor, PATRAN.[59,60] Having generated the geometric model it is then possible to create the finite element model. In the case where geometry is created within CATIA, this is taken via a data exchange mechanism e.g., Initial Graphic Exchange Specification (IGES), through to the finite element analysis package NASTRAN. In the Structures Unit it is often preferable to use PATRAN as the pre and post processor to the

NASTRAN finite element analysis package.[61] In terms of the current design methodology being an infrastructure for processing knowledge, this area represents a significant knowledge bottleneck. This is because within the current design methodology the finite element model generation process is largely dependent on the operator's expertise. It was appreciated that this specialist knowledge could be processed more efficiently if a degree automated guidance where to be given in the areas of model idealisation and the determination of element density.

As can be seen from Figure 3.1, once the finite element model has been generated, it goes forward for analysis and structural optimisation. It is not considered appropriate or practical in the context of this thesis to discuss any of the concepts relating to the discipline of structural optimisation. However, there are excellent texts readily available by Morris, Arora and Vanderplaats which the author recommends.[62,63,64]

BAe MA&A utilises the ECLIPSE structural optimisation system. This system will optimise the sizes of elements in a finite element model in order to reduce the stress levels within the idealised model. In addition, there is the facility within the system to perform individual component optimisation. For instance, an analysis of the model may reveal that a component within the model has an unacceptably high stress concentration level. This component (e.g. a lug) may be taken out of the model and have the optimisation process performed on it separately in order to achieve the necessary stress reduction. It should be noted that where a component is taken out of context to be shape optimised it will be necessary to run the optimisation procedure for the entire model again. The structural optimisation process requires that the tacit knowledge of the structural engineer be applied iteratively as he or she works towards an exceptable solution. Clearly, in the context of the structural optimisation process being part of a larger knowledge management process there is potential for this area of the process to become another knowledge bottleneck. That is, where a relatively inexperienced engineer is performing the optimisation it is likely that the operation will take longer than when performed by an experienced member of staff. Whilst, it is appreciated that it is not possible to mitigate entirely for staff experience, it is considered that the potential for a knowledge bottleneck can be alleviated through the introduction of appropriate AI technology to provide a degree of automation to this iterative process.

The ECLIPSE structural optimisation system is capable of performing a comprehensive post processing of results from whatever type of analysis/optimisation has been performed. Neutral files can be produced directly for subsequent processing in PATRAN.

As can be seen from Figure 3.1, the results from the analysis and optimisation processes are fed to the design and aerodynamics disciplines. This information will obviously be complex e.g., giving details of sizes and the lay-up of composites and mass stiffness. In the case of both the design and aerodynamics disciplines the results output by the finite element analysis (FEA) model may be further analysed and optimised. The results from these additional analysis and optimisation processes may

be an end in themselves or conversely be the starting point for a further iteration in order to better define the real structure i.e., the entire cycle is repeated.

The information concerning finalised designs is stored in a variety of places, ranging from CAD database, microfiche to paper drawings and calculations. However, while this information may readily be accessed it is not structured in such a manner as to provide any form of explicit guidance with respect to future designs. The lack of structured access to information concerning finalised designs has the effect of slowing down the design process. It is important to appreciate that for most companies, the central issue is not creating organisational knowledge. Instead, it is identifying how to more effectively capture and share the knowledge that already exists within the organisation, but which is locked within a department, a division, or even within the minds of individual staff. In the context of BAe MA&A, it is considered that the introduction of AI technology capable of structuring past design information in such a format that it can readily be queried and the most relevant information accessed appeared to offer considerable potential with respect to better leveraging organisational knowledge in relation to finalised designs.

This chapter has introduced BAe MA&A's current design methodology and highlighted the limitations of the methodology in the context of it being a mechanism for managing knowledge. In addition, this chapter has drawn attention to the said limitations as being areas where AI technology could be employed to enhance the effective processing of knowledge within the methodology and thereby eleviate the limitations discussed. To summarise, three specific areas were identified where AI technology could be employed to facilitate the processing of product knowledge:

1. To provide a focal point for the inter-disciplinary interactions necessary when deriving the initial real structure.
2. To provide guidance with respect to the design analysis process. Specifically, the building, optimising and analysing the FEA model.
3. To provide a means to enable previous design cases to assist with the derivation of the real structure.

As discussed in the introduction, this chapter has taken a strategic view of the current design methodology and identified how it is desired to channel company product knowledge. Chapter 5, moves away from the strategic view and steps closer to the operational level of abstraction to identify the individual mechanisms that will facilitate the processing of company product knowledge. Specifically, Chapter 5 introduces a new design methodology for BAe MA&A, incorporating the appropriate AI technology necessary to facilitate an improved knowledge management process. However, before discussing this new methodology, Chapter 4 now discusses AI technology for design.

# Chapter 4

# Artificial Intelligence Technology for Design

This chapter is divided into four main sections. The first section discusses the AI technology applied within this study. The second section provides an overview of other AI technology researched as being possibly of use within the study but were subsquently rejected. In each of these sections, the reasons for selecting and rejecting the AI technology discussed are given. The third section of this chapter discusses the important role that the object oriented methodology has to play with respect to developing AI based applications. The final section of this chapter leads on from this discussion to introduce the frame-based approached to structuring knowledge, this AI technique possesses many of the attributes of the object oriented methodology.

## 4.1. Artificial Intelligence Methodologies Employed within the Research Project

### 4.1.1. Expert Systems

Whilst expert systems contain the majority of the features of knowledge-based systems (KBS) they are not 'strictly speaking' one of the same. They differ in one significant respect. A traditional expert system tends to focus on just one knowledge domain, while a KBS system is able to embrace more than one knowledge domain. By inference this implies that a KBS is able to support more than one rule base. In addition, it should also be noted that with knowledge-based engineering (KBE) systems which are advanced KBS (e.g., the ICAD™ system) there is also an implicit understanding of product geometry.[65] This facilitates the generation of a geometric model which conforms to rules and relationships stored in the knowledge base. Clearly, there is a need to make the distinction between KBS and expert systems.

#### 4.1.1.1. Defining Expert Systems

As indicated above, expert systems and KBS share many features in common. The definition of expert systems given below holds true for KBS allowing for the differences indicated above.

An expert system is a computer program which behaves like a human expert in some useful way. Expert systems solve difficult problems using application-domain knowledge and problem solving skills. They are tools for solving problems that normally require a human expert. Expert systems, as mentioned previously, are designed to solve problems in a single discipline or domain of knowledge. Unlike traditional programs that use algorithms, expert systems solve problems using deductive reasoning. As a result, expert systems can solve problems that are unstructured. Expert systems have a wide range of capability, they can be used to assist, advise, diagnose, analyse, consult and categorise.

The importance of expert systems as tools is increasing. The concept of using knowledge to solve problems where algorithms cannot be found is proving fruitful.

The cost of computing power no longer limits the application areas where they may be used.

Expert systems allow anyone with expertise to outline processes used to solve a problem. Once done, the outline can be used to generate a series of questions which lead a non-expert through the solution of similar problems. Most expertise is refined so that that it can be applied repeatedly and this is what expert systems exploit. It is important to note that the expert is never replaced, but access to expertise is extended. From a management perspective, when considering the intoduction of expert systems, it is this aspect which must be emphasised in order to smooth the path for their introduction in the workplace i.e., alleviating fears that the installed expert systems will be a substitute for staff.

In building expert systems, knowledge of experts must be captured and stored, such that it can be used to make deductions. The biggest problem in building most expert systems, is not developing the program, but distilling the information from the expert into codified form. The knowledge illicitation process can be a difficult process as the 'knowledge engineer' needs to be able to differentiate between what is useful knowledge and what is just data. For example, in the aircraft industry, the knowledge engineer can readily access vast quantities of data but identifying the 'nuggets' of knowledge is a significant challenge.

Expert systems have limitations. They require regular monitoring as the system will tend to degrade quickly at the boundary of the knowledge domain. Clearly, as expert systems are only computer programs they cannot be expected to make an analogy, use common sense or apply intuition.

#### 4.1.1.2. Requirements of an Expert System Language

The requirements of a programming language capable of developing an expert system with are as follows:

- The language should support data-driven structures.
- The language must support symbolic processing i.e., should be able to do formal reasoning with symbols.
- The language must support list processing i.e., it should be possible to build symbolic structures in lists.
- The language must support recursion.

### 4.1.2. Knowledge-Based Systems (KBS)

Knowledge-based systems (KBS) while still a relatively new technology in the computer aided engineering (CAE) marketplace have already proven to yield significant benefits. Implemented correctly they permit the user to complete work in minutes which previously took days or even weeks to complete. In addition, to improving productivity times, their capability to bring together engineering knowledge from disparate departments within a company has enabled significant reductions in product life cycles to be made.

The most important long term benefit that KBS offer a company is that they possess the capability to capture the engineering experise of the organisation and as such true product knowledge.

The principle component of a KBS is the knowledge-base or rule base. This can be thought of as a framework in which all the design characteristics and relationships of the product are encapsulated. The creation of the knowledge base is technically a demanding and time consuming task. Clearly, the person(s) creating the knowledge base must have intimate knowledge of the product's characteristics.

The knowledge base has a comprehensive structure for evaluating input parameters, and is capable of optimising these parameters against the design requirements. KBS structured correctly are flexible enough to prioritise the rules within the knowledge base (e.g. cost against material against environment) and make decisions based on the fitness for purpose of a design.[66]

### 4.1.2.1. The Architecture of a Knowledge-Based System

Figure 4.1 illustrates the basic archititure of a KBS. The principle components of a KBS are:

- The knowledge base
- The inference mechanism
- The working memory
- The knowledge acquisition module
- The explanation module
- The user interface module

The operation of each of these components is now outlined below.

**Knowledge Base**

The knowledge base of any KBS is the most important part of the system. The reason being that the performance of the system depends on the knowledge that it contains. The knowledge base may contain a range of different types of knowledge; facts about objects; actions and events; cause and effect relations; performance knowledge i.e., knowledge about how to do things; meta knowledge i.e., knowledge about what is known. The representation of the knowledge-base is dependent on the implementation of the system.

**The Inference Mechanism**

The inference mechanism or engine controls the KBS's activity by manipulating the current process using the knowledge base. The inference mechanism contains no domain knowledge. The capabilities of an inference mechanism include; the selection of appropriate actions to modify the context of the current process, for example, inferring new facts; replacing the current state of a fact by a new fact; matching knowledge base patterns with the current state of the problem; implementing actions.[67]

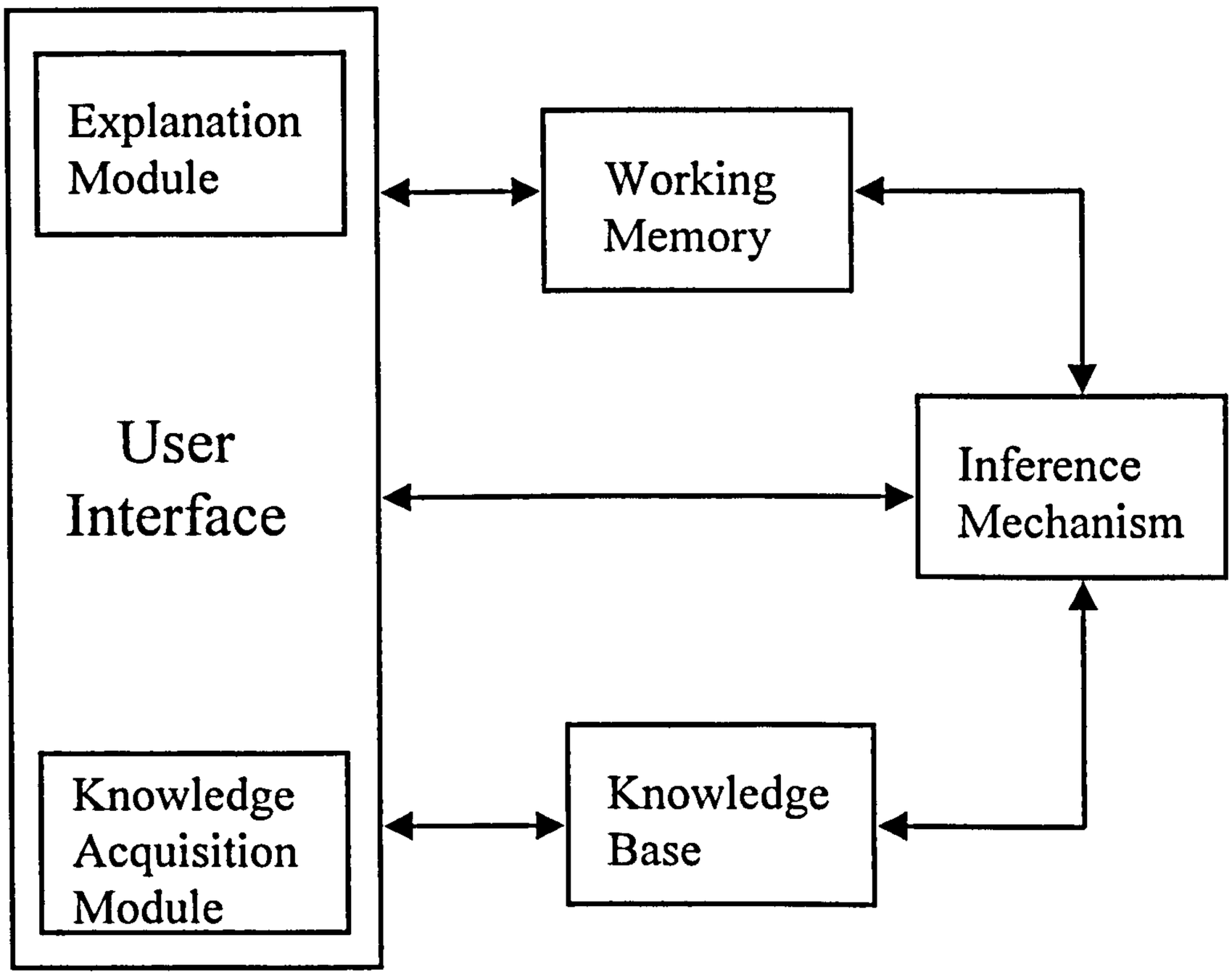



Figure 4.1, Architecture of a Knowledge-Based System

**The Working Memory**

The working memory contains facts that reflect the current state of the problem solution. The working memory is initially empty but builds up dynamically as the problem solution progresses.

**The Knowledge Acquisition Module**

The knowledge acquisition module provides an interface between the system and the expert(s). In addition, the module will assist in developing the knowledge base and structuring facts in the working memory for a particular knowledge domain.

**The Explanation Module**

The purpose of the explanation module is to provide the user with explanations of why certain inferences where made and conclusions reached. The form and detail of the explanation will vary according to the knowledge domain.[68] However, a good system module will be able to supply the user with a complete audit trail.

**The User Interface Module**

The user interface module provides an interface between the user and the system. The user interface will usually provide the user with some form of command language guide to operating the system. The user interface module will designate the form that input must be entered in and where appropriate transfer user input into a form that is useable by the system.[69]

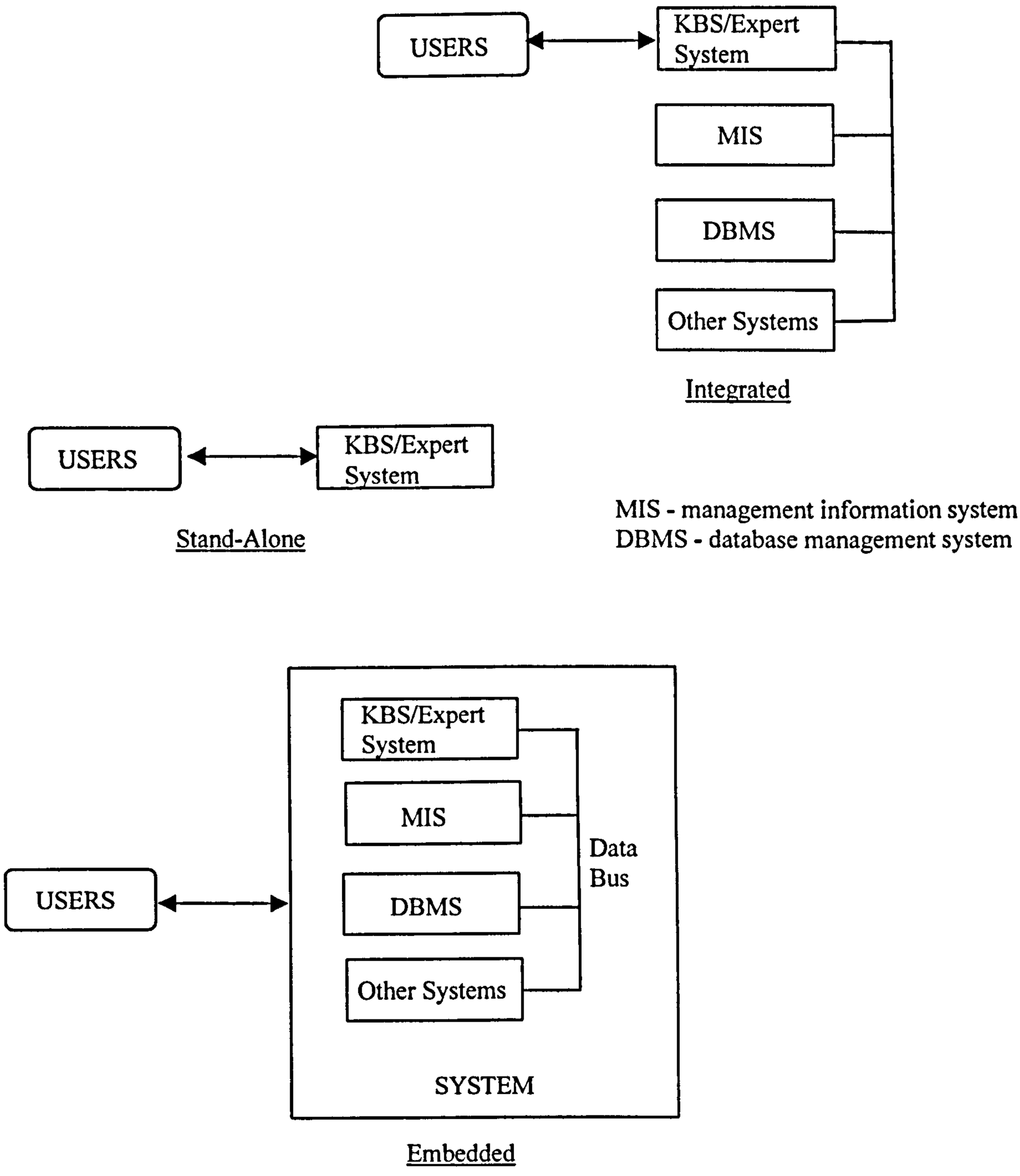



Figure 4.2, KBS and Expert System Modes of Operation

#### 4.1.2.2. Knowledge-Based and Expert System Modes of Interaction with other Computer Systems

There are three modes of operation or ways in which an expert system or KBS can perform its chosen role, stand-alone,integrated or embedded.[70] These three modes of KBS/expert system interaction are illustrated in Figure 4.2

**Stand-Alone**

In the stand-alone configuration the KBS/expert system interacts with the user only.

**Integrated**
The operation of the KBS/expert system involves the exchange of data between the KBS/expert system and other systems as well as dialogue with the user.

**Embedded**
In this mode of operation the KBS/expert system is completely absorbed within an information system. The user does not interact directly with the KBS/expert system but indirectly through the user interface of the host. With an embedded KBS/expert system it is quite possible for the user to be oblivious to the existence of the KBS/expert system.

### 4.1.3. Case-Based Reasoning

Case-based reasoning (CBR) is an AI technique which utilises past design experience in order to assist with the solution of current design problems.[71] With CBR, design cases are stored within a database of previous cases. Rather than deriving design solutions from a domain knowledge base composed solely of rules and relationships, previous design cases are examined and similarities brought to the users attention. The system can then highlight to the user where significant differences between the stored cases and the current case exist. This then provides a good starting point or basis for deriving a suitable solution for the current design problem.

The idea of reasoning from relevant passed design cases is appealing in as much that it corresponds to the process that an expert might use to solve problems. In addition, a case base has several advantages over a conventional production production rule knowledge base. These advantages are summarised below.[72]

**Knowledge Acquisition**
When it comes to knowledge acqisition, design cases are much more memorable to the expert than specific rules and relationships. A further advantage with respect to the knowledge acquisition process is that the collecting of knowledge and encoding it into a series of IF-THEN rules is difficult. This is because to a large extent experts rely on their experience rather than a rigorous set of rules and relationships. An additional complication in setting-up rules is it is necessary to trace interactions between rules to ensure that they chain together properly and that contradictions are eliminated.

**Learning from Experience**
A significant advantage CBR has over a conventional rule base is that the case base is continually growing and improving as a design tool as new cases are entered into the system. In this way, the case-based system has progressively more capacity to deal with new design cases as they are presented to the system. A rule base however, requires that additional rules and facts be entered into the rule base independently of the design problems. Broadly speaking, a case base attempts to apply 'experience' whilst a rule base attempts to encapsulate 'expertise' and apply that.[73]

**Memory**

The structure of rule base systems do not have the capability to memorise the design cases they encounter. Instead, for each design case the system has to work through the rule base from the start no matter how many times before the system has dealt with a similar case.

**Robustness**

For a rule-based system to be of use, the rules it incorporates must encapsulate the design problems the system is likely to encounter. Clearly, if a problem does not match any of the rules the system will not be able to solve the problem. From this view point rule-based systems are not robust.

#### 4.1.3.1. The Relationship between Rule Base and Case Base Technology

It should not be thought that case-based and rule-based systems are mutually exclusive, in fact they are complementary systems. It is quite concievable that one could interogate a CBR system and fail to find a satisfactory solution to the current design problem. In such circumstances there is a requirement for a rule-based system to step in and offer guidance with respect to how the design process should proceed. In addition, in hybrid systems, rule bases are often used to guide the user to an appropriate case(s).

#### 4.1.3.2. The Structure of Design Cases

A design case must contain a list of features which represent the nature of that case. As far as possible these features should have some element of uniqueness which will set one case apart from another e.g., possibly material specification or manufacturing processes employed. It should be noted that when the number of cases within a case base becomes large, retrieval normally relies on *indexes*. An index is an auxiliary data structure that provides a direct mapping from each specific feature of interest to cases having that feature.[74]

Individual design cases may contain information relating to all the disciplines involved in the design process e.g., design, manufacturing, aerodynamics. As such, there is a need to structure this information in a manner that reflects the disciplines involved i.e., a modular structure, and yet still enabling the easy retrieval of design cases. The frame-based approach to structuring knowledge and object oriented programming which are discussed in section 4.3 provide such a means of structuring design cases.

#### 4.1.3.3. Storage Requirements and Retrieval Cost

The case-base developer must be aware of storage requirements and the retrieval cost of design cases. Although memory is becomming increasingly plentiful, at any given time memory is a finite resource which needs to be traded off against other possible uses. Consequently, one significant issue is the amount of memory required to store cases. As the quality of solution retrieved by a case-based reasoner is expected to be monotonically increasing with the number of distinct, relevant cases available to it,

one wants to accommodate as many cases as possible. Clearly, design cases must be defined concisely and stored efficiently if computer memory is not to be a problem.

It should be noted that retrieval cost is a function of the number of cases examined and the effort required to determine their relevance to the current situation. Both the number of cases examined and the cost of deciding among them can be large in practical case bases.[75] Obviously, there is a requirement to make the comparison between design cases as easy and as simple as possible.

#### 4.1.3.4 Problems Involved in Developing a Case Base

Before closing this discussion about CBR it is worth summarising those problems that must be faced in the development of a case base:

- The relevant features of a design case must be identified.
- The design cases must be entered into a computer in a consistent and concise manner.
- The representation of interdependency among features of a design case must be addressed.
- The definition of similarity between cases must be addressed i.e., what constitutes similarity?
- The ambiguities in similarity that arise will have to be addressed
- What methodology will be employed if no case is found to be similar?
- The most suitable method for storage and retrieval of design cases nust be determined

In summary, it was decided to utilise KBS and CBR in colaboration as they appeared, after investigation, to be be the most suitable technologies to facilitate the capture and subsequent re-use of product knowledge. Whilst KBSs are capable of focusing on generic rules that relate to a problem, CBR is able to focus on specific instances of similar problems to the current one. As indicated in section 4.1.1, the two AI technologies are complimentary and appeared to be the most suitable techniques around which to build a new design methodology.

## 4.2. Artificial Intelligence Technology Considered for Application within the Research Project

This section briefly outlines the salient points of genetic algorithms and neural networks. During the initial stages of this study both these disciplines were considered for use within the development of the methodology but were subsequently rejected for the reasons given.

### 4.2.1. Genetic Algorithms

It should be borne in mind that all design studies are a progressive series of compomises were the constraints and variables acting on the design process are optimised. As such, it was deemed appropriate to investigate the potential of genetic algorithms in the context of this study.

Genetic algorithms are a search and optimisation method which derives its name from a loose analogy with a genetic change in a population of individuals. Based on the Darwinian principle of evolution, genetic algorithms are designed to mimic evolutionary selection. Most genetic algorithms assume a feature-based representation of instances and events. Genetic algorithms employ a distinctive set of knowledge structures or patterns. Each pattern specifies the presence (or absence) of some features. Patterns also have an associated weight, sometimes called the pattern's fitness, that summarises its performance on past experiences.[76]

The optimisation method employed by genetic algorithms is fundamentally different to that employed by gradient-based optimisation methods. With the genetic algorithm approach, a population of candidate designs is evaluated at each iteration. The probability of each candidate design being reproduced, and being present in the next generation (iteration) depends on its fitness value. The fitness value relates directly to the value of the objective function i.e., how similar it is to the objective function. Progress toward the optimum is achieved by the intoduction of new candidate designs through the application of operators such as crossover and mutation.

Each individual candidate design is described by a binary string. This is basically a coded listing of the values of the design variables. These binary strings can be considered to be analogous to biological chromosomes, with genes for different features of the candidate designs.

The purpose of the crossover operator is to enable two individual candidate designs to swap part of their binary descriptive strings. This enables new candidate designs to be created, being based on the combination of parts of old strings.

Mutation operators are necessary because, even though reproduction and cross over effectively search and recombine to produce new candidate designs, occasionally they may become over zealous and lose some potentially useful genetic material (1's or 0's at particular string locations). In artificial genetic systems, the mutation operators protects the process against irrecoverable loss. In a simple genetic algorithm, mutation is the occasional (with small probability) random alteration of a string position.

Many genetic algorithms employ a variety of special operators in conjunction with reproduction, crossover and mutation, usually introduced for specific applications. The standard genetic algorithm described here is that based on the algorithm by Goldberg and is limited to these three fundamental operators.[77]

The advantage that genetic algorithms present over gradient based optimisers which are and continue to be used extensively in aircraft design, is that they are able to operate on discontinuous functions. In addition, they permit variation in the number of design variables during optimisation. Reported applications of generic algorithms to date have included pattern recognition, control system optimisation, layout scheduling, the parametric design of aircraft, and the preliminary design of turbine engines.[78]

This study has revealed several instances where genetic algorithms have been used with some success in the area of design optimisation. It would appear however, that while genetic algorithms are good at determining the area of the design space where a global optimum exists they do not guarantee convergence to the optimum.[79,80] As such, it was deemed likely that to be effective genetic algorithms would have to be supported by a more convergent optimising mechanism.

Turning attention now to the aims of the of this study i.e., to develop a methodology that would enable product knowledge to be captured and subsequently delivered to the points of need. In the early stages of the research project the author did not discount the use of genetic algorithms in the development of a new design methodology. However, it was apparent as the research project evolved that genetic algorithms as a tool focus on the generation of new designs and in the context of developing a methodology that facilitates the capture of product knowledge genetic algorithms contribute little. As is acknowledged in the conclusions to this thesis there is certainly scope to embrace genetic algorithms if the methodology presented in subsequent chapters were to be built upon. However, it was decided that their immediate use with in this study was not appropriate.

### 4.2.2. Neural Networks

Neural networks are a form of pattern recogniser. A neural network system is taught to recognise certain features or aspects of a design. This is achieved by training the system by 'showing' it a number of example designs. When trained, a neural network system will present the user with the example in memory that most closely matches a new design presented to it, a level of confidence of the match will usually also be given. The advantage of a neural network over other options is that they are computationally very efficient once set up. In addition, neural networks are insensitive to numerical instabilities and convergence difficulties typically associated with computational processes.[81]

In the context of this research project it was recognised at an early stage that there would not be available sufficient training data to make the use of neural networks a viable option. As such, it is not proposed to discuss neural networks in any further detail here, but instead, refer the interested reader to references 82 and 83.

## 4.3. The Object Oriented Methodology

The adoption of the object oriented methodology has been one of the reasons for the advancement of artificial intelligence (AI) technology in recent years. It is proposed here to briefly outline this important methodology.

The object oriented approach permits the definition of an object. Objects can represent physical and non-physical entities. Objects are usually members of an object class whose definition defines attributes and operations of class members. The attributes and operations may be inherited from one or more super-classes so that a class definition need merely set out the difference between that class and its super classes.[84] Each object contains knowledge about itself and its interactions with other

objects. The process of creating an object instance from a class definition is called instantiation.

In the context of programming, object oriented programming languages differ considerably from conventional procedural languages such as Pascal, Basic, C or FORTRAN. The principle feature of a procedural language is that procedural languages require the programmer to describe the procedures for manipulating data within the program. Clearly, a procedural language based program can be no more flexible than the procedures it contains. Object oriented languages, on the other hand, are of modular design, with a minimal requirement for procedures, making them ideal for handling relatively unstructured problems i.e., such as the problems presented in this study.

## 4.4. The Frame-Based Approach to Structuring Knowledge

Frames are an AI tool for structuring knowledge. They embrace many of the attributes of the object oriented methodology outlined above. In frame theory, the important relationships, properties and concepts are placed into a common data structure called a frame. A frame can consist of objects and facts about a situation, or procedures on what to do when a given situation is encountered.

The knowledge associated with a frame is contained in structures called slots. The slots may store factual and procedural knowledge associated with the frame. The slots may be an end in themselves or they may lead to additional sub-frames. With frame networks, at the top of the structure is found general, common-sense knowledge, while at the bottom of the structure is found very specific knowledge. This is due to the hierarchical structure of frame networks. The hierarchical structure of frame networks is illustrated in Figure 4.3.

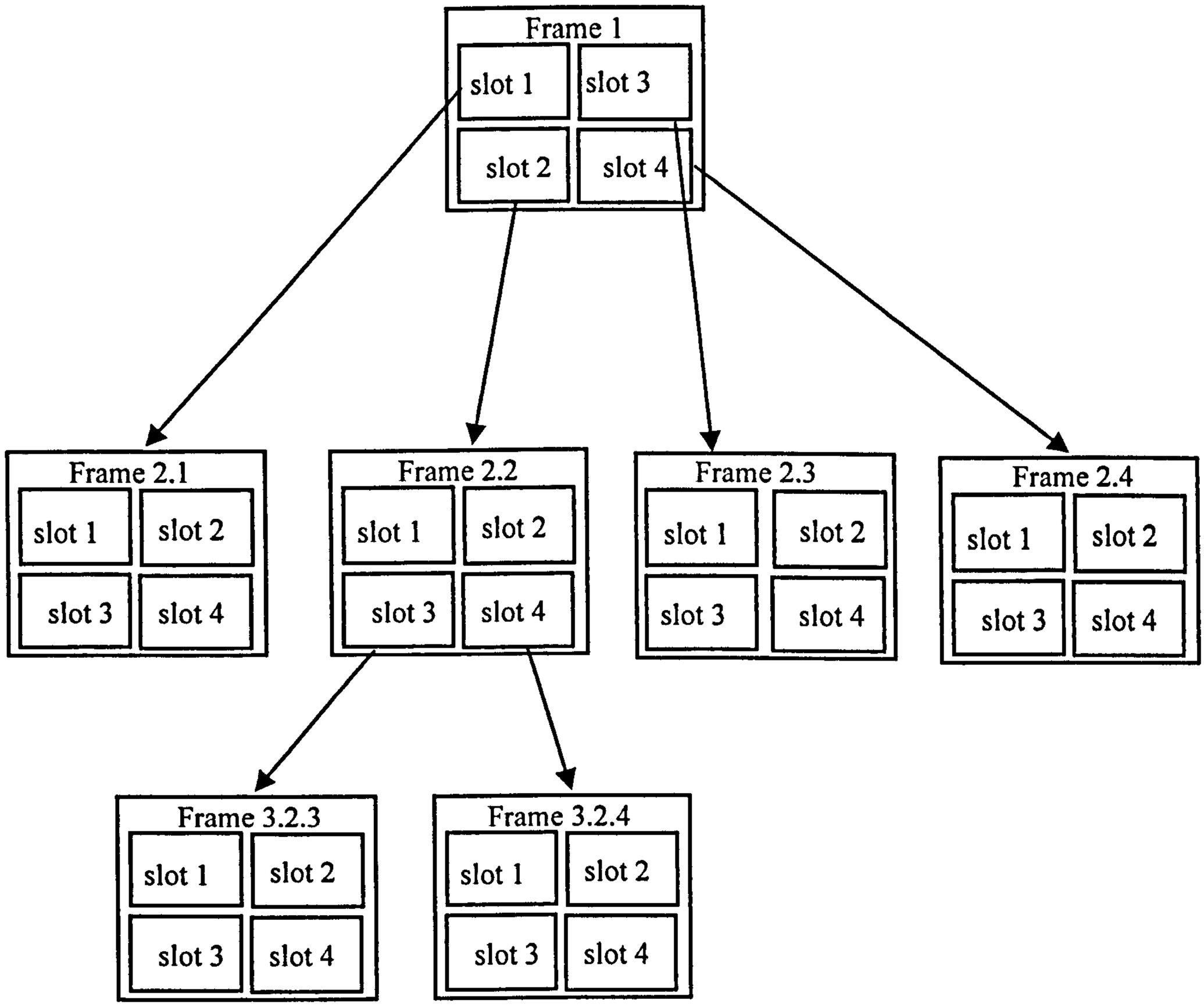



Figure 4.3, Frame Network Hierarchy

The object oriented nature of frames makes them ideal for the encapsulation of information from a wide range of disciplines, such as has been encountered during this study. It is considered that the frame-based approach is the most appropriate method of describing cases in the CBR environment.

Having discussed the AI technology which it was decided to utilise within the new design methodology, Chapter 5 now introduces this methodology and discusses the location of the KBS and CBR disciplines within it.

# Chapter 5

# New Design Methodology

As indicated in Chapter 1, traditional design methods do not foster knowledge retention and re-use. There is a need for a new approach to the design process. This chapter, through the application of artificial intelligence (AI) technology presents a new design methodology whose purpose is to manage knowledge in a manner which is superior to traditional approaches and thereby overcome the current limitations of traditional design methods with respect to knowledge capture, retention and subsequent re-use.

As indicated in Chapter 3, the development of the new design methodology in the context of it being a mechanism for managing knowledge is a three phased process. Chapter 3 took the strategic view, identifying the types of knowledge present and how it would be desirable for this knowledge to be channelled. This chapter, now moves to the next lower level of abstraction and identifies the most appropriate mechanisms and processes to facilitate the effective management of company knowledge. That is, providing for its capture and subsequent re-use.

It is important to point out that while this chapter introduces a new design methodology for BAe MA&A. The methodology presented is generic and as such it has the potential to be applied to other aircraft manufacturers or even be adapted for other industries which have a similar underlying design process.

This chapter is divided into two sections. The first section provides a methodology overview. Whilst, the second section identifies the specific areas within this methodology where the detailed research of this study was focused.

## 5.1. Methodology Overview

Figure 5.1 illustrates the principle component parts of the new design methodology. It can be seen that Figure 5.1 is a closed system. The reason for this is that aircraft design is not a simple design process which has a definitive start and finish. Aircraft design, is a dynamic process which is constantly evolving as new design tools and techniques become available. For example. Consider the Tornado. When the Tornado was introduced in the early 1980s, the first variant was the GR1. The Tornado since its introduction has been developed to support a range of roles, ranging from reconnaissance to air defence (the ADV variant). The basic ground attack version is now the GR4 variant.

In addition, when the Tornado was designed in the 1970s estimates for component life expectancy were made with contempory analysis software. As improved analysis tools have become available, combined with in-use data it is possible to make improved life expectancy estimates for the various parts of the aircraft. It is because of this evolutionary process that Figure 5.1 is represented as a closed system. It is now proposed to describe the components of the new design

methodology in a logical manner. Thus, the description will start with the real structure definition (1).

As with the current methodology, the new methodology defines the real structure by drawing information from the three discipline areas as initially indicated in Chapter 3 i.e., non-structural mass distribution (2), aerodynamics and aeroelastic constraint definition (3), and design constraint definition (4). However, a knowledge-based system (KBS) (5) now provides the focal point for the range of inter-disciplinary interactions that must take place in order to derive the real structure. It is important to point out that by placing a KBS at this strategic point there is the potential to draw information from all disciplines that have a vested interest in the design process. In the context of knowledge management, the KBS provides staff with ready access to organisational knowledge. As alluded to in Chapter 1, access to organisational knowledge has become more critical to achieving and sustaining competitive advantage than the knowledge that individual staff members possess. The KBS facilitates knowledge sharing and enables staff to access critical information quickly. In addition, the KBS allows staff to readily gain an appreciation of the impact of their decisions on other disciplines. This helps the design process to move away from the traditional 'over-the-wall' approach to design.

By providing the focal point for the definition of the real structure as it does i.e., Figure 5.1, 1, 2, 3, 4, the KBS is capable of providing a range of different roles e.g., focusing on aerodynamic requirements, or possibly non-structural aspects of the design process. However, in the context of this study it was decided that the purpose of the KBS should be to identify the 'best structure' from the manufacturing and structures perspective, providing the designer with efficient and effective assistance at the conceptual design stage. The KBS is closely linked to a case base (7); this emphasises the point made in section 4.1.3.1 of Chapter 4, that CBR and rule base systems are complimentary to each other. This linkage between the two disciplines enables the design methodology as a mechanism for managing knowledge to direct both heuristic knowledge and previous design information to the designer in a readily useable format.

While the operation of the KBS (5) and case base (7) is explained in detail in Chapter 7, it is appropriate here to provide an overview of what these important components of the methodology do and how they support the knowledge management process. The KBS takes user input and applies a series of generic rules in order to identify the 'best structure' in the context of the manufacturing and structures disciplines. The KBS operates from the top-level, whereby the system asks the user what the component in question that it is desired to design offers to the aircraft. On the basis of the inputs entered by the user, the knowledge base identifies the design drivers that act on the design process. Supported by further interaction with the user, the KBS goes on to identify the best structure. This best structure is determined by the ranking of structural types in descending order of preference by a structures rule base and a manufacturing rule base. When the preferred structural types of each rule base are in agreement then this structure is presented to the user as being the best structure. In terms of the management of knowledge, the KBS mechanism draws design knowledge into the system at a relatively high-level of abstraction and progressively

directs this knowledge until a specific solution is derived. This solution may then be taken forward in the methodology.

Turning attention to the case base (7). The case base provides access to past design cases. This mechanism provides a repository for design experience and

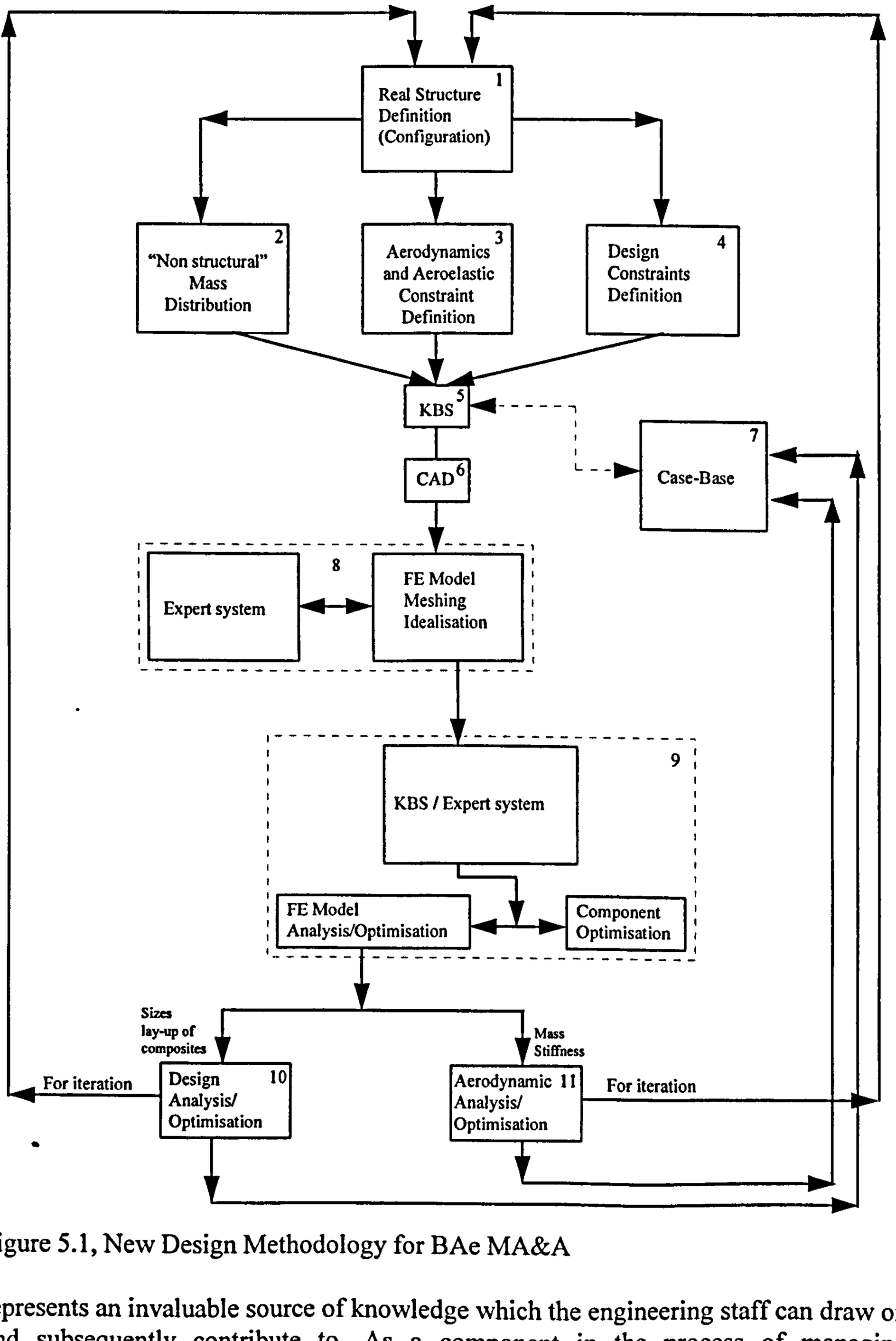



Figure 5.1, New Design Methodology for BAe MA&A

represents an invaluable source of knowledge which the engineering staff can draw on and subsequently contribute to. As a component in the process of managing

knowledge the case base represents a method to transfer knowledge to related disciplines. The nature of the case base structure permits the quantity and quality of knowledge stored to improve as the design cycle iterates i.e., as new cases are entered into the case base. The case base facilitates experience sharing. This capability facilitates the multi-discipline sharing of experiences and thus improving the core skills of the engineering staff. This is achieved through the placement of the case base in the knowledge management architecture represented by the new design methodology. The case base improves skills by providing two types of knowledge; product specific knowledge and skill specific knowledge. Product specific knowledge is a set of knowledge relating to a specific product. It can easily be represented in documents such as engineering drawings and performance specifications. On the other hand, skill specific knowledge is general knowledge and skills needed to develop products.

It can be seen from Figure 5.1 that the output from the KBS supports the creation of the geometric model. The geometry is generated within the computer aided design (CAD) (6) environment. Once the geometric model is completed, as with the current design methodology, it is then possible to generate the finite element model (8). As can be seen from Figure 5.1, the new design methodology provides an expert system to provide guidance with respect to finite element model idealisation and element density decision making.

Once a satisfactory meshed finite element analysis (FEA) model has been generated it is then ready for analysis and optimisation processes to be performed on it. In order to assist with these processes the new methodology provides an additional expert system to guide the FEA model to analysis or component optimisation or both as deemed appropriate (9).

The expert system capability placed in the finite element modelling analysis and optimisation area of the new methodology aims to relieve the potential for knowledge bottlenecks. This issue was initially raised in Chapter 3 when a strategic view of the current design methodology was taken with respect to the methodology being an efficient system for processing company product knowledge.

As with the current design methodology, the output from the FEA model is fed to the design (10) and aerodynamics disciplines (11). For both these disciplines the results output by the FEA model may be further analysed and optimised. The results output by the design and aerodynamics disciplines may take one of two possible routes. Firstly, the results may be fed back to the real structure definition (1). Thus, causing the entire cycle to be repeated until a satisfactory definition is found. Secondly, when a satisfactory solution is found the design and aerodynamic information is fed to the case base (7) in order for this information to assist with the solving of future design problems. It is important for the reader to appreciate that the output from the FEA model may be fed to any interested discipline. However, for the sake of discretion focus has been placed on the two mentioned.

To summarise, the new methodology provides appropriate AI technology in three specific areas and thus facilitates the capture and subsequent application of product knowledge:

- A KBS (5) provides a focal point for the inter-disciplinary interactions necessary when defining the real structure.
- A case base (7) provides the facility whereby previous design cases can be stored and used to assist with the derivation of the real structure i.e., the solution of future design problems.
- Expert system technology is placed in the design analysis arena to provide both guidance and a degree of automation in this area i.e., specifically, to assist with the building (8), optimising and analysing (9) the FEA model.

These three areas combine to facilitate the effective flow of knowledge through the new design methodology. The AI technology presented provides access to important knowledge assets i.e., residing in functional disciplines and past design experience. This knowledge is channelled in an effective manner to provide the potential to reduce product development times.

## 5.2. Focus for Detailed Research

It was appreciated that in order to further advance the methodology it would not be possible to develop all the above three areas at once, but rather it was necessary to develop the methodology in manageable stages. As such, it was decided that the initial focus of the project should be on the KBS link between the real structure definition and CAD. This link also embraces the case base requirements. The reason for selecting this area as the starting point is that the interactions in this area facilitate the definition of the conceptual design. It has been well documented that decisions made at the conceptual design stage commit approximately 80% of the down stream production costs.[85] Thus, by providing the capability to effectively manipulate product knowledge in this area there is the potential to yield the most benefit to BAe MA&A.

Having decided to focus the development of the methodology in the conceptual design arena, it was necessary that those mechanisms employed within the methodology should be validated. As such, a software prototype was developed so that the methodology could be appraised in the context of its ability to capture and subsequently utilise product knowledge. In order to test the prototype it was necessary to select an appropriate structure to drive through the system. It was decided that an aircraft foreplane represented such a structure.

This chapter has provided an overview of a new design methodology for BAe MA&A. The areas were it is intended to employ AI technology to facilitate the capture and re-use of product knowledge have been highlighted; the specific technologies being KBS and CBR. The conceptual design arena has been identified as the area where the subsequent research of this study was focused. Chapter 6 now discusses background knowledge in design and in particular focuses on the role of CBR in the aircraft design environment and the influence of research and development (R&D) on the design process.

# Chapter 6

# Background knowledge in Design

This chapter is composed of two sections. The first explores the important role that research and development (R&D) and company business drivers have to play in the design process and indicates how this role was encapsulated within the methodology. The second section, discusses how traditional CBR is limited in the context of it being suitable for application in the aircraft design environment. The section leads on to introduce the necessary concepts that in the view of the author make the CBR technology a viable tool to be applied within the aircraft design environment.

## 6.1. The Role of Research and Development and the Company's Business Drivers in the Design Process

An area often overlooked in design studies is the very important role that R&D and the company business drivers have to play in the design process. The introduction of new manufacturing processes and design principles are achieved primarily through expenditure in R&D. This R&D may be performed in-house or be bought-in from universities or comparable institutions. It is important to note that if the role of R&D is ignored during the development of an AI support system for the design process then the methodology employed will be flawed.

When considering the role of R&D it is necessary to be aware of the relationship between R&D and production. In many instances a concept may be deemed highly desirable from the R&D standpoint but in reality prove to be very difficult to translate into the production environment. If conflict arises between a new concept being supported by an R&D innovation and the current manufacturing/production capabilities then a conflict resolving decision has to be made.

There are three alternative solutions available for resolving such a conflict. The first solution is to ignore the contribution offered by R&D and just utilise standard manufacturing capability, thereby 'playing safe' and thus ensuring a manufacturable product. The second alternative equates to the opposite extreme, whereby the desirable R&D option is accepted in an unqualified manner. This leads to a situation where the product cannot be manufactured to the required standard. Clearly, neither of these solutions represent practical alternatives.

The third solution takes the 'middle road' between these extremes. Here, it is acknowledged that a wide range of options exist that are not currently within the scope of current manufacturing capability. However, it is acknowledged that the R&D skills will be present to create this capability. In addition, it is appreciated that there will be cost and time factors associated with developing new manufacturing/production competencies which management will need to consider. This third approach is the one adopted in the software prototype discussed in this study. Here an R&D 'flag' is used to highlight the state of current research and to indicate, where appropriate, that further research is required.

Apart from the impact that R&D makes on the design process it is necessary to consider the role of company business drivers. Clearly, all companies have to appraise their business strategies on a regular basis. As part of this appraisal process management should identify those core competencies that the company possesses i.e., those competencies which provide the company with a sustainable competitive advantage, and those core competencies which it is desirable for the company to acquire.

Once management have identified those core competencies that the company is lacking measures can be taken to obtain them in order to maintain, and in the longer term, enhance the company's competitive advantage. The acquisition of new competencies may necessitate the company 'by-passing' the conventional processes of building up company knowledge. For example, when an executive decision is made that a certain component will be manufactured using a particular process in order for the company to enhance its manufacturing capability in a particular branch of production.

In the context of developing the methodology and subsequently the prototype it was necessary to distinguish between those competencies which the company possesses and those which it is trying to enhance or acquire. As such, the resulting prototype has the capability to distinguish between proven technology at hand and that which is being developed.

Chapter 7 describes the method employed by the new methodology for highlighting the impact made by R&D and company business drivers. Whilst, Chapter 9 shows through example, how this method is implemented within the software prototype.

## 6.2. Case Based Reasoning in Aircraft Design

Applying CBR in the military aircraft design arena poses some unique problems. In most environments where CBR is applied there are lots of 'recipes' for the case reasoner to interrogate. To illustrate this point consider the CBR system Battle Planner which can interrogate a database of 600 historical battles.[86] Similarly the CBR system BROADWAY advises on the selection of automobiles to purchase; clearly, the quantity of cases that can be attributed to this knowledge domain is considerable.[87] In comparison, the number of cases that reside in the military aircraft knowledge domain is relatively small. In addition, whilst there are only a few aircraft cases, these cases are huge and possess attributes which are peculiar solely to the aircraft industry. For instance, aircraft are designed to have different roles, e.g., air superiority, ground attack, and as such have a wide range of different technologies applied to them. Further, the duration of the formation of the design case may range anywhere between twelve and twenty years and each case is likely to be a radical departure from what has gone before.

Assembling the design knowledge that relates to a particular case is difficult for a variety of reasons. Designers don't design an entire aircraft anymore but rather, design

teams focus on particular areas of the design. Further, the same designers that designed the initial aircraft are unlikely to work on the subsequent variants. Indeed, design teams may not stay together thoughout the initial development. Due to the length of the gestation period of aircraft design projects it is likely that a designer will not work on many more than two aircraft projects in his or her entire career. Clearly, in such circumstances it is quite conceivable that the product knowledge that relates to a particular design case is at the very least widely dispersed throughout the company or possibly no longer within the company i.e., designers may have moved out of the company or may have even retired. This aspect is markedly different from the knowledge domains to which many case base reasoning systems are applied. Domains where CBR is currently employed tend to be very compact and the associated knowledge is relatively easy to obtain, thus making cases easy to assemble in comparatively short periods of time. Consider CHEF which operates in the food domain and outputs recipes.[88] The knowledge domain encompasses all forms of culinary skills and is generally extremely easy to access. In comparison to the sprawling nature of military aircraft design, cases in the food domain are light weight, compact and quick to assemble which some would say makes them ideal for CBR. However, it is argued here that to evolve into a really useful technique CBR must address knowledge domains that are both challenging and ungainly.

CBR should facilitate adaption. In the aircraft design arena the application of adaption poses problems not confronted by conventional CBR systems. In the majority of existing CBR systems where adaption is offered it is possible to adapt cases and validate the quality of the adaption to high degrees of certainty; usually through applying rules to the adapted cases or performing simple tests. For instance, CHEF's adapted cases can easily be validated to ascertain the quality of the adaption i.e., are the recipes edible and does the food taste good? However, the aircraft industry is somewhat different. Here, if one adapts a case which is based on a real aircraft e.g., European Fighter Aircraft (EFA), then problems arise with respect to the validation of the adapted case. That is, to ensure that the adapted case is of the same standard as the known design there are a range of validation procedures that may be and in some circumstances must be applied to the adapted aircraft design case if it is to be deemed acceptable. These tests may range from mathematical calculations, FEA and possibly flight testing. The high financial costs and time involved with respect to such involved analysis would detract from any potential advantage that CBR may offer to such an extent that there would be no advantage in using a CBR system. Clearly, for CBR to be of benefit in the aircraft industry it is desirable to provide a system that does not require that every case derived through adaption be tested to the extremes described.

As indicated above a significant difference between military aircraft design cases and cases from most other knowledge domains is the potential huge size of military aircraft design cases. As such, the indexing of cases presents a problem. In conventional case base reasoning systems only those aspects of a case are indexed which will differentiate one case from another. In addition, it is assumed that the user will also have the appropriate input information to trigger these indexes. In the military aircraft knowledge domain the number of differentiating indexes is potentially huge and it is very unlikely that the user will have sufficient information to trigger all the indexes. It is therefore likely that the 'most suitable' case will be

retrieved from the case base on the basis of relatively scant information. Therefore in the majority of instances many aspects of a retrieved case will not have been compared with the input triggers or specifications and are just presented with the rest of the case i.e., the case is retrieved on the basis of a best match with known information. The failing here is that those aspects of a retrieved case for which it is not possible to perform any comparison with the specifications may be crucial with respect to deciding whether a case is appropriate or not. For instance, the user may not have any information with respect to structural type i.e., be it carbon fibre, traditional stressed skin or super plastic formed, diffusion bonded (SPF/DB) titanium. Clearly, this information is vital in the context of selecting one case as opposed to another.

To summarise, a case base reasoner operating in the military aircraft knowledge domain will have to handle a small number of large cases which are very different from one another. Further, if case adaption is to be applied usefully there needs to be a means of minimising the validation procedures and yet permit adapted cases to provide guidance which is comparable to validated real cases. In addition, as aircraft cases have the potential to be extremely large any indexing procedure employed must be suplimented by additional methods that take into account those aspects of a case that are not embraced by the indexing procedure employed.

With the scenario described, it was apparent that a methodology was required that would permit an aircraft design case retrieved from the case base as being the 'best case' to be examined in greater depth. As such the case base reasoner developed within this study, and outlined in detail in Chapter 7, ranks cases in descending order of preference with respect to the input specifications i.e., best to worst match. The case base then operates a form of jury system whereby the 'best case' is required to defend itself against the next best case. What this process attempts to achieve is to focus attention on those aspects of the best case which were not triggered by the input of the known desired specifications. These aspects are now brought to the fore and the case presents arguments for these aspects. In addition, where practical the case presents arguments against alternative choices.

The case base developed in this study also offers case adaption. In the situation where an aspect of a case has not been defended successfully it is possible for the equivalent component in the next 'best case' to be substituted. In this manner using substitution techniques a case may be adapted. Adapted cases may then be presented to the user as tentative design guides. It is important to note that adapted cases are segregated in the case base from real cases. This is because adapted cases are not as creditable as fully validated cases e.g., the EFA or the Experimental Aircraft Programme (EAP). Chapter 7, which outlines the details of the methodology employed within the conceptual design arena provides a fuller explanation of how adapted cases may be utilised in the case base and a method is proposed whereby they may approach a level of validation.

This Chapter has discussed the important role that R&D and company business drivers have to play within the design process. Attention has been drawn to the need to embrace these components of the design process within the development of the methodology and subsequently the software prototype. In addition, the chapter has

focused on the unique problems that the operation of a CBR system in the military aircraft knowledge domain throws up. The chapter has provided an explanation of how the CBR technology has been expanded within this study to embrace the sprawling nature of aircraft design cases. Chapter 7 now discusses in detail the development and operation of the new design methodology operating in the conceptual design arena. Here it will be explain how CBR and KBS operate together in harmony to facilitate the capture, storage and re-use of product knowledge.

# Chapter 7

# Intelligent Conceptual Engineering System (ICES) Methodology Architecture

This is the principle chapter of this thesis. *This chapter documents the contribution to knowledge made during this research project.* This chapter discusses in detailed the component parts of the new design methodology operating in the conceptual design arena. The chapter is composed of four sections. The first section provides an overview of the logic behind the ICES methodology. The second section discusses the KBS, the third discusses the case base operation and the final section focuses on the interaction between the KBS and the case base. Having decided to start the detailed development of the methodology in the conceptual design arena it was deemed appropriate to name this portion of the methodology. As such, the methodology developed in this area is hence forth referred to as the Intelligent Conceptual Engineering System (ICES). The ICES methodology architecture described in this chapter describes the component parts of the system, their function and how they interface with one another. In appendix A the ICES methodology architecture described here is documented in greater rigor using the Integration Definition Zero (IDEF0) system analysis tool.

The contribution to knowledge documented in this chapter lies in two areas. Firstly, the design driver ranking technique which assigns justifiable numerical weightings to design drivers acting on the design process in conjunction with the approach employed to ensure consensus between a manufacturing rule base and a structures rule base with respect to deriving a 'best structure'; specifically, this approach employs the concept of utilising secondary rules. Secondly, this chapter introduces a highly novel new concept to case base reasoning called the 'jury technique'.

Before discussing ICES in depth it is appropriate here to place the system in context with respect to it being a mechanism for processing and managing company product knowledge. As indicated in Chapter 3, the development of the new methodology as a process for managing knowledge is a three phased process; Chapter 3 took a strategic view identifying the types of knowledge and how most effectively this knowledge should be channelled; Chapter 5 identified the appropriate technology to facilitate the channelling of knowledge in the desired manner i.e., KBS and CBR technology. This chapter now moves to the lowest level of abstraction and discusses in detail how these technologies are implemented within ICES.

## 7.1. ICES Methodology Overview

ICES was developed with two modes of operation in mind. This is because two possible scenarios are envisaged where the user may wish to interact with the system. In the first scenario the user has no explicit specifications for the structure under consideration and as such would wish to interrogate the KBS in order to determine the 'best structure'. Having determined the best structure via the KBS, ICES then tells the

user which design case residing in the case base represents the best match to the structure recommended by the KBS. In the second scenario, the user has a set of specifications, from which he or she wishes to know whether any design case residing in the case base represents a close match to the specifications. The user enters the specifications into the system. The system then compares the specifications to the design cases in the case base and presents the user with the best match. Figure 7.1 illustrates the underlying logic supporting the ICES methodology. The remainder of this section will now step through the operation of the ICES methodology as presented in Figure 7.1 in order to provide the reader with an overview of the system.

As can be seen from Figure 7.1, the ICES KBS first requires the user to enter the component of interest i.e., that which it proposed to design, and determine the design drivers acting on the design process. The design drivers are determined via the operation of a meta rule base as described in section 7.2.1. These design drivers are assigned numerical weightings. This is achieved the through the application of the design driver ranking technique (DDRT) as outlined in section 7.2.2. This technique represents part of the first contribution to knowledge made in this study. The DDRT addresses the difficult problem of assigning ranked importance weightings to design drivers acting on the design process.

Having entered the component that it required to design and determined the design drivers acting on the system, the ICES KBS presents a range of structural types to a manufacturing rule base and a structures rule base e.g., traditional stress skin, carbon fibre composite (CFC) and super plastic formed (SPF) titanium structures. These rule bases, through the firing of a series of rules assign a numerical score to each of the structures under consideration. The structures are then ranked in descending order of preference by each rule base. The preferred structures of each rule base are then brought together and a match is attempted, see Figure 7.1. If a match is achieved i.e., the preferred structure of both the manufacturing and structures rule bases are in agreement, then this structure is presented to the user. This aspect of the ICES KBS is described in detail in section 7.2.4. However, if as is likely, the preferred structure of the manufacturing and structures rule bases are not in agreement then the ICES KBS permits the user to access and fire a series of secondary rules. The purpose of these secondary rules is to enable a match to be obtained between the preferred structure of the structures rule base and the preferred structure of the manufacturing rule base. The concept of the secondary rules as applied to this study represents a further contribution to knowledge. The operation of the secondary rule is discussed in detail in section 7.2.4.2. It can be seen from Figure 7.1, that once a match is obtained through the application of secondary rules, or otherwise, the system presents the case(s) in the case base that possess this particular structure. This interaction between the ICES KBS and the case base is discussed in section 7.4.

Turning attention now to the ICES case base. As indicated above and in Figure 7.1, it is assumed that the user has a set of specifications to which he or she wishes to know whether a design case residing in the case base matches. The user enters the specifications into the case base and through the application of search mechanisms, the case that most closely matches the specifications is presented to the user. As can be seen from Figure 7.1, the ICES case base now requires the best case to justify or

defend itself with respect to the next best case. The defence presented by the best case highlights the positive aspects of the case and also the negative points of other cases

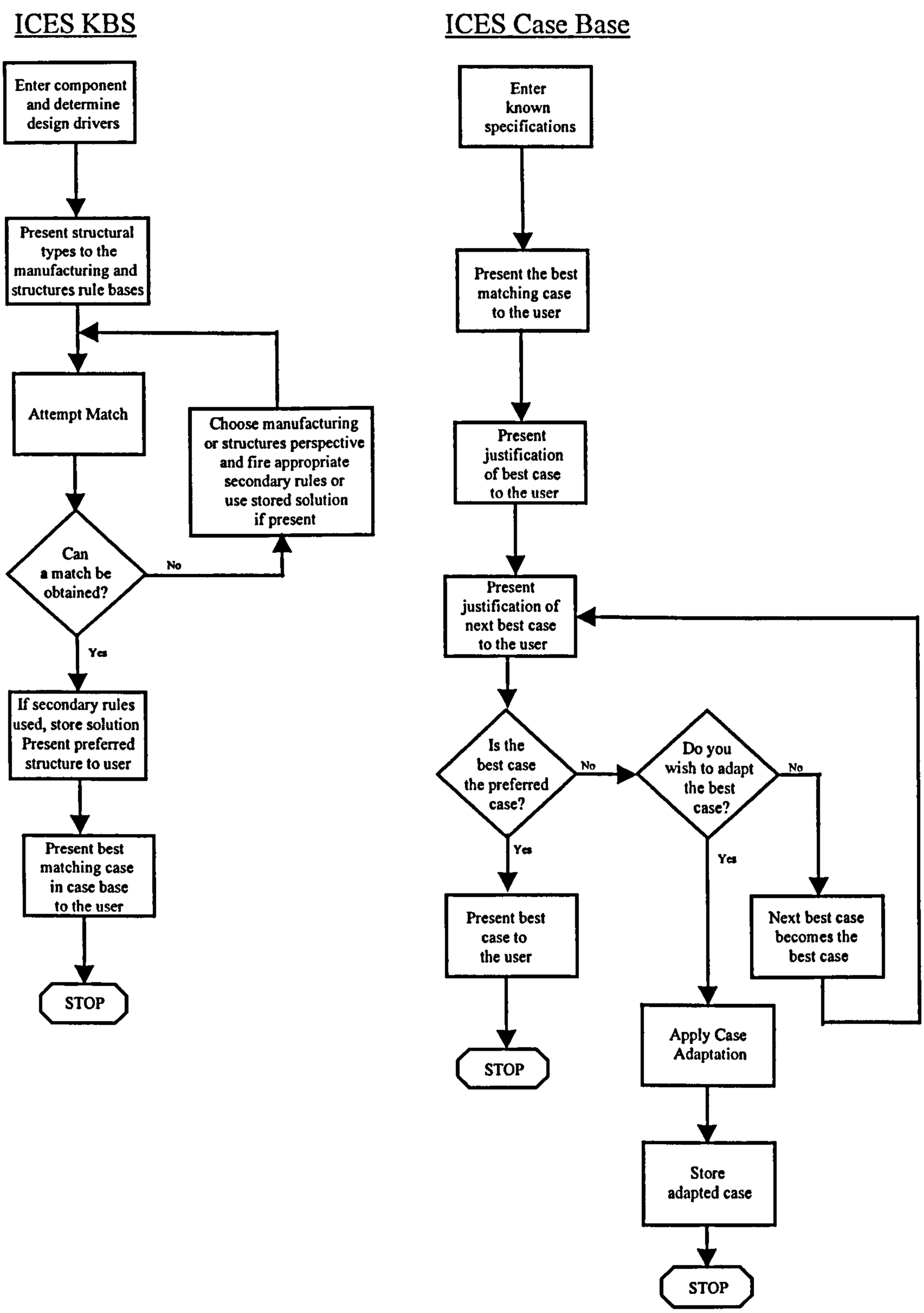



Figure 7.1, Overview of the ICES Methodology

residing in the case base. This defence is aimed primarily at the next best case which the user may also access. The next best case will, as with the best case, put forward a defence as to why it is superior to the best case and all other cases in the case base.

The user acting as the 'jury' decides to accept of reject the defence put up by the best case. If the best case is rejected then the next best case becomes the best case. This new best case must now defend itself against a new next best case. As can be seen from Figure 7.1, the process continues until the current best case defends itself successfully. This concept of requiring design cases to defend themselves is called the 'jury technique' and is completely new to the case base reasoning methodology and represents a further very significant contribution to knowledge made by this study. The jury technique is discussed in detail in section 7.3.3.

It can be seen from Figure 7.1, that in the jury loop there is the possibility to adapt a case in the case base. If the user is dissatisfied with the cases residing in the case base as individual entities but is interested in aspects of different cases then the ICES case base permits the user to employ case adaptation. This aspect of the ICES case base is described in section 7.3.4.

## 7.2. KBS Operation

The KBS supports the following three rule bases:

- Meta rule base
- Manufacturing rule base
- Structures rule base

In addition, the KBS provides a method for ranking the design drivers acting on the design process in descending order of importance. This method is referred to as the design driver ranking technique (DDRT). The operation of these components of the KBS are now outlined below.

### 7.2.1. The Meta Rule Base

It is important to note that while an aircraft foreplane was the structure driven through the ICES prototype, it was the intention during the development stage of the prototype that the system remain generic as far as possible and as such have the potential to support other structures apart from the foreplane.

The KBS takes a top-down perspective. This starts with the meta rule base which presents the user with a collection of generic terms common to aircraft operation. The user selects those representing the principle operational characteristic features required by the design requirement e.g., agility, supersonic performance, subsonic performance, durability. In addition to the generic terms relating directly to aircraft performance, additional generic terms are provided that relate to the broader company view of design e.g., cost, R&D, and the role of core competencies. These features may also be selected by the user.

Having made the necessary selection, the generic terms are matched against a set of generic meta rules. These identify the fundamental relationship between the generic terms common to aircraft, those relating to the broader company view, and what is physically required to achieve the generic terms. The output from the meta rule base is a generic list of design drivers which act on the design process. These design drivers

are given a numerical weighting. This numerical weighting is derived through the DDRT which is outlined in the following section. In order to illustrate the meta rule base consider the design of a foreplane where the principle design characteristic as a whole is agility. Inside the meta rule base the following rules relating to the generic agility term are applied.

- Good instantaneous turn rate requires low wing loading. Wing loading is minimised by reducing *wing weight*.
- Good sustained turn rate requires low wing loading. Wing loading is minimised by reducing *wing weight*.
- Good beyond visual range turning capability necessitates a minimum turn radius performed possibly at supersonic speed. Thus there is a requirement for *high structural strength.*
- Specific excess power (SEP) is derived from excess thrust x velocity/weight. *Minimising weight* increases the SEP
- Control surfaces capable of high pass rates implies the requirement for *low torque* forces.

The meta rule base outputs the information, in the form of design drivers. To achieve agility the principle design drivers are:

- minimum weight
- high structural strength
- low torque

Clearly, in order to be fully comprehensive, additional design drivers need to be taken into account which cross reference with other rule sets.

As indicated in section 5.1 of Chapter 5, the meta rule base as part of the KBS, draws design knowledge into the system at a high level of abstraction. That is, through requiring the user to identify the operational characteristic features, the design drivers acting on the design problem are identified in a logical manner. In terms of a knowledge management process the meta rule base is in effect driving the design process forward to the next level of detail.

### 7.2.2. <u>The Design Driver Ranking Technique</u>

As indicated above the design drivers obtained through the operation of the meta rule base are assigned a numerical weighting. As commented upon in Chapter 8, the first prototype developed in this study did not use derived numeric weightings i.e., the numerical weightings were arbitrary numbers. However, for the final ICES prototype it was essential that an appropriate method of assigning weightings to the design drivers be found. The use of 'historical weights' was the first method of assigning weights to be explored. The concept here was to examine available aircraft documentation and interview BAe MA&A engineers and on the basis of historical information assign numerical weightings to the design drivers. Whilst this idea has merit, it was considered to be flawed for two reasons. Firstly, it was considered unlikely that the examination of aircraft documentation would yield much historical evidence on which to base the design driver weights. Secondly, if BAe MA&A

engineers were to be interviewed there was no means to ensure that the correlation of the information obtained would not lead to numerical weightings that were not unfairly bias. That is, it would not be practical interview all engineers who have a valid input to make and as such there was no means to select the most appropriate ones. Even if it were feasible to interview all BAe MA&A engineers it is conceivable that the weightings obtained would tend to have a bias due to the way British Aerospace operates engineering projects i.e., the weightings obtained would not be truly generic. Another consideration which relates to the system to effectively manage knowledge is that it was essential that the mechanism utilised for assigning weightings to the design drivers must be assured of being maintained as current and accurate. It appeared that the accuracy of weightings based on histororical data could have a tendancy to degrade quickly. In addition, there was no means to be assured of their status i.e., current relevance and accuracy. A further issue that is also worthy of consideration is the relative difficulty of transferring design driver weightings based on historical information into a software environment where the weights can be readily updated.

Having explored and rejected the possibility of using of historical weightings it was appreciated that there was a requirement for a new technique to be developed for determining the numerical weightings of design drivers. This new technique needed to be practical, relatively easy to use, and generic i.e., making it free from bias. The design driver ranking technique (DDRT) developed in this study and now described, outlines such a method, it is both generic and readily facilitates the derivation of numerical ranked weightings. In addition, as verified by the creation of the ICES prototype which is discussed in Chapter 9 the DDRT can easily be transferred to a software environment.

The DDRT ranks the design drivers determined via the meta rule base in order of importance using a matrix based approach. The approach applies two distinct operations to the design drivers. Firstly, the design drivers are compared against each other with respect to how they impact on one another and a numerical score is attributed to this relationship. The mean average of these impaction scores for each design driver is then calculated. The level of impaction is determined using the order of magnitude scoring system as outlined in the following section. Secondly, each design driver is considered with respect to how sensitive it is to change i.e., how strongly a change in a specific driver causes a physical change in the design. The design driver importance ratings are determined by multiplying the mean strength of interaction score by the design drivers' sensitivity score. The DDRT can be summarised as follows:

*Mean strength of interaction between design drivers x Speed at which a change in the design driver impacts on a design = Importance Rating*

It is important to note that this technique is not trying to obtain 'exactitudes' with respect to inter-design driver relationships but rather good justifiable 'ball park' relationships. The following subsections now outline in detail the operation of the design driver ranking technique.

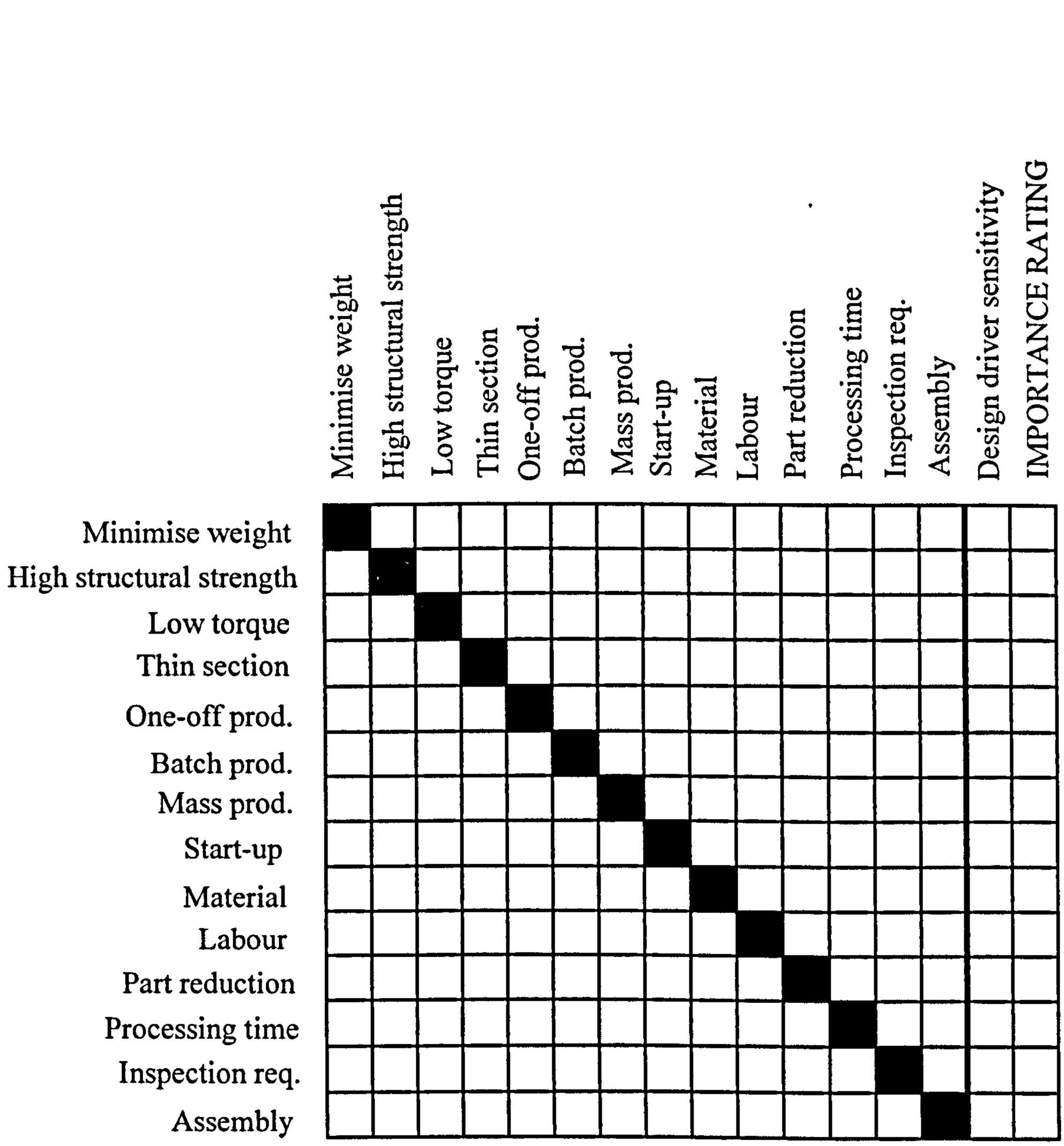



Figure 7.2, Design Driver Matrix

### 7.2.2.1. The Design Driver Matrix

As can be seen from Figure 7.2, the design driver matrix is divided into two distinct sections, separated by a thick vertical line. The first, and largest section, places the design drivers along the vertical and horizontal axes. Working down the vertical axis, each design driver on this axis is compared against each design driver on the horizontal axis. The comparison takes the form of how the design drivers on the vertical axis impact on the design drivers on the horizontal axis. To illustrate the impact of one design driver on another consider how the minimise weight design driver impacts on the other design drivers:

- High structural strength - the requirement for minimal weight impacts on the requirement for high structural strength by dictating to a large extent the geometry of the structure.
- Low torque - the requirement for minimal weight impacts on the low torque design driver in a similar manner i.e., a solid structure could possibly be precluded. By making the structure lighter the effort required to move the structure is reduced.

- Thin section - the requirement for minimal weight impacts on the thin section design driver in as much as it would probably be impractical to have a solid thin section.
- One-off production - the requirement for one-off production may exclude some methods of manufacture which may facilitate the minimisation of weight.
- Mass production and batch production - the requirement to minimise weight impacts on the mass and batch production design drivers in no obvious manner.
- Start-up - the requirement to minimise weight impacts on start-up costs in that high quality tooling may be required to ensure that weight is kept to a minimum.
- Materials - the requirement to minimise weight impacts strongly on the material design driver i.e., determining which material is selected.
- Labour - the requirement to minimise weight has only the slightest impact on labour requirements. It can be argued that certain materials will be excluded in preference to the more exotic in order to save weight. Thus, the labour skills required to work with these exotic materials will be more specialised. Clearly, the relationship between the minimise weight and labour design drivers is not particularly strong but an argument can be presented for it.
- Part reduction - the requirement to minimise weight has a direct impact on part reduction. Considerable effort is spent in trying to combine parts in order to achieve overall weight reduction.
- Processing time - the requirement to minimise weight does not impact on processing time.
- Inspection requirements - the requirement to minimise weight does not impact on inspection requirements.
- Assembly - the requirement to minimise weight may impact on assembly in as much as there will be efforts made to reduce the fastener count.

Clearly, it is difficult for the user to determine how strong the impact of one design driver is on another. With this in mind, the user is assisted in deciding the degree of impact between design drivers by the provision of a series of generic order of magnitude choices. Their purpose is to guide the user in deciding how strongly one design driver impacts on another. Each of the generic order of magnitude choices is assigned a numerical weighting and a generic textual description. The numerical weightings start at 0 where there is no relationship between design drivers; the weightings increase progressively to a maximum score of 1 where a design driver makes a very large impact on another design driver. The range of order of magnitude choices is listed below.

**Order of Magnitude Relationships**

| Symbol | Relationship | Score |
|---|---|---|
| A<<B | A has no impact on B | 0 |
| A-<B | A has a very small impact on B | 0.2 |
| A~<B | A has a relatively small impact on B | 0.4 |
| A>~B | A has a significant impact on B | 0.6 |
| A>-B | A has a relatively large impact on B | 0.8 |
| A>>B | A has a very large impact on B | 1 |

Before continuing it is appropriate here to discuss the requirement for the range of order of magnitude choices to be transitive. The basis for such a requirement is best illustrated by considering the consequences of intransitive preferences. Suppose, therefore, the following preferences where expressed $a_1 < a_2$, $a_2 < a_3$ and $a_3 < a_1$ among three options $a_1$, $a_2$, and $a_3$ where < means 'preferred to' and $a_1$, $a_2$, and $a_3$ are different order of magnitude choices. The assertion of preference rules out equivalence between any pair of options. As such, the expressed preferences reveal that there is some actual difference in value (no matter how small) between the two options in each case. Considering now the behaviour implications of these expressed preferences. Taking the preference $a_1 < a_2$, there is by implication a 'price', say x, that would have to be paid to move from a position of accepting option $a_1$ to one where option $a_2$ is acceptable. Let y and z denote the corresponding 'prices' for switching from $a_2$ to $a_3$ and from $a_3$ to $a_1$ respectively. Suppose it is now necessary to accept option $a_1$. By virtue of the expressed preference $a_1 < a_2$, and the above discussion, x must be paid to move from option $a_1$ to option $a_2$, and pay y to move from option $a_2$ to $a_3$. Repeating the argument once again, z must be paid to move from option $a_3$ to $a_1$. Thus, the total payment would be x+y+z in order to return to the starting position $a_1$. Willingness to act on the basis of intransitive preferences is thus seen to be equivalent to a willingness to suffer unnecessarily the certain loss of something to which one attaches positive value. This is regarded as inherently in consistent behaviour. Thus, preferences should conform to the following:

$$\text{If } a_1 < a_2 \text{ and } a_2 < a_3, \text{ then } a_1 < a_3$$

and should be understood in the following sense; to avoid expressing preferences whose behaviour implications are such as to lead to the certain loss of something of value, then care must be taken to ensure that preferences fit together in a transitive manner[89]. In this respect the order of magnitude choices listed above are considered to be transitive and thereby consistent.

It is appropriate at this juncture to discuss how the numerical values used in the order of magnitude choices were derived. Clearly, how the range of order of magnitude choices are assigned will have a direct bearing on the output of the DDRT i.e., the importance ratings. In this study, the order of magnitude choices move through the order of magnitude values with evenly spaced numerical step scores with no sudden changes in the steps taken. The reason for selecting evenly spaced numerical steps is that this permits the user to express gradually increasing or decreasing levels of impact with respect to one design driver on another with a corresponding increase or decrease in numerical score. Conversely it would not be sensible to have changes in level of impact out of step with the assigned numerical scores.

Having expressed the reason why the numerical scores are assigned to the order of magnitude choices as they are in this study, it is important for the reader to appreciate that there is no reason why other range formats cannot be used. It is conceivable that management may wish to emphasis different aspects of the listed order of magnitude choices; possibly placing extra emphasis on those design drivers which impact strongly on one another. It is considered that the DDRT could be

enhanced with research into the area of applying different range formats. A possible starting point could be to apply a range of standard series formalisms to the DDRT e.g., logarithmic, binomial and Taylor's series, and monitor the impact on the importance ratings. It is conceivable that management could generate a series of order of magnitude ranges; each one aimed at putting emphasis on different types of design driver impaction.

The order of magnitude technique permits the combining of different orders of magnitude impaction. Consider a situation where it is known that a design driver has an impact on another design driver but the degree of impact is not known because of a lack of supporting data. As such, it is conceivable that the impact of one design driver on another may be supported by one or more order of magnitude choices. Where the impact of one design driver on another covers a range of levels of order of magnitude then the mean average is taken as the level of impaction. To facilitate the understanding of how the order of magnitude technique could be applied a couple of examples are know given.

Firstly, consider a situation where design driver A makes no impact on design driver B, from the orders of magnitude above, this relationship scores 0. However, in some circumstances design driver A will make a very small impact on B; again from the list of orders of magnitude relationships above this relationship scores 0.2. Taking the mean average of these levels of impaction gives a final value of 0.1.

Secondly, one might have a situation where the impact of design driver A on design driver B may vary. For instance, it might be the case that usually 'A makes a very small impact on B', giving 0.2, but in certain circumstances 'A makes a significant impact on B', giving 0.4. As with the previous example, the mean average of these levels of order of magnitude is taken giving an impaction score of 0.3.

A significant feature of this order of magnitude technique is that it is possible to determine the level of inter-design driver impact where the impact of one design driver on another design driver may vary depending on the specific structure in question. To illustrate this facet, consider a structure which may be manufactured from either super plastic formed diffusion bonded (SPF/DB) titanium or carbon fibre composite (CFC). Clearly, it is likely that the design drivers will impact on each other in different ways depending on which structure is under consideration. The order of magnitude technique enables the points of view relating to each structural type to be combined in the same manner as described in the two examples given previously.

The order of magnitude concept provides efficient integration of quantitative and qualitative knowledge in the expression and solution of engineering problems. Order of magnitude ideas have already been used in AI. In an earlier effort, order of magnitude concepts have been examined as a means for algebraic simplification.[90] More recently, in a diagnostic system for digital circuits.[91,92] However, it is believed there use within the DDRT represents a novel use of the concept.

Using either strict or heuristic interpretation, the knowledge acquisition process can be greatly enhanced by the order of magnitude methodology. Instead of asking the

expert to give hard numbers and exact relations a system can ask the expert questions of the type:

- Is A much larger than B?
- Which of the relating A<<B, A-<B, A~<B could hold between A and B?

In the context of this study it is not considered appropriate to discuss the concepts behind the order of magnitude formalism further but direct the interested reader to reference 93. In conclusion, it is considered that those aspects of the order of magnitude formalisation used within this study have illustrated the capability of the technique to bridge the gap between traditional qualitative reasoning and full quantitative reasoning.

### 7.2.2.2. Design Driver Sensitivity

The smaller portion of the design driver matrix (to the right of that part of the matrix which is dedicated to design driver interaction) focuses on design driver sensitivity, see Figure 7.2. As indicated in section 7.2.2, the sensitivity relates to how quickly a change in a design driver will cause a physical change in the design. Specifically, numerical weightings are assigned to design drivers depending on by how much they can be changed with respect to how strongly the change in the design driver acts on the design. For instance, it could be argued that the physical characteristics of a design are fairly insensitive to changes in the number of people working on a design project (labour design driver) in terms of how quickly any change will actually alter the integrity of the design. However, changes in the material used (material design driver) will probably impact quickly on the design. The range of design driver sensitivity and associated weighting are as follows:

| **Sensitivity** | **Weighting** |
|---|---|
| The design driver is 'totally' sensitive to change | 1.0 |
| The design driver is very sensitive to change | 0.8 |
| The design driver is moderately sensitive to change | 0.6 |
| The design driver is slightly sensitive to change | 0.4 |
| The design driver is completely insensitive to change | 0.2 |

### 7.2.2.3. Design Driver Ranking Technique Formula

The relationships present in the DDRT outlined above are summarised in the following formula.

$$g_j = \sqrt{\left(\sum_{i=1}^{n} \frac{a_{ij}}{n}\right)} \times b_j$$

where:

$g_j$ is the importance rating of design driver j. The importance rating is given in the range 0 to 1.

$a_{ij}$ is the strength of interaction of design driver i with j where ij = 1 to n.

The sum of the impaction scores is divided by the number of design driver interactions n to give a mean impaction score. It is considered appropriate to take the mean average of the impaction scores as it provides a means towards encapsulating in a single figure the relative importance of a single design driver as viewed by a designer(s) at a particular instance.

$b_j$ is the sensitivity of design driver j i.e., how quickly the design driver impacts on the design. The sensitivity value $b_j$ is multiplied by the mean impaction score to give the overall importance rating for the design driver.

The reason for the square root is to ensure that the importance rating output falls in the range 0 to 1. Failure to employ a square root would mean that some of the output importance ratings would be an order of magnitude smaller than the desired range of 0 to 1. As the formula is an empirical formula the use of a square root is acceptable in this instance.

In the equation given above it was considered that if the design driver was particularly sensitive to change then this should be highlighted with respect to the final importance rating. Hence, the design drivers are multiplied by a relatively higher numerical score the more sensitive the design driver is to change. Clearly, it is quite possible that management may wish to highlight those design drivers that are the most stable i.e., least sensitive. As such it is conceivable that the sensitivity weighting scores could be inverted whereby a design driver which is completely insensitive to change multiplied by the highest sensitivity weighting.

It is important to note that this formula is an empirical formula and should not be considered in any other context e.g., a measure of distance.

#### 7.2.2.4. Design Driver Matrix Update

With respect to developing the prototype, it was necessary for ICES to permit the design driver matrix to be up-dated. By getting a wide range of potential users to declare how they believe the design drivers impact on each other and by taking the mean average of these results a more accurate/stable importance rating for each design driver is obtained. In addition, by drawing input from a large number of users, this component of ICES is receptive to input representative of the wide range of disciplines residing in the organisation and thereby enables ICES as a whole to be a more effective mechanism for processing knowledge. The concept of the design driver matrix update is illustrated in Chapter 9 which discusses the ICES software prototype implementation.

### 7.2.3. The Role of Research and Development

It was acknowledged in section 6.1 of Chapter 6 that research and development (R&D) has an important role to play in determining the 'best structure'. It was highlighted that the degree of R&D input may be due to a natural design progression or possibly be the result of strategic input by senior management. In terms of

developing a software prototype is was considered that the most appropriate method of determining the degree of R&D input required was to ask the user to select one of the following levels of R&D.

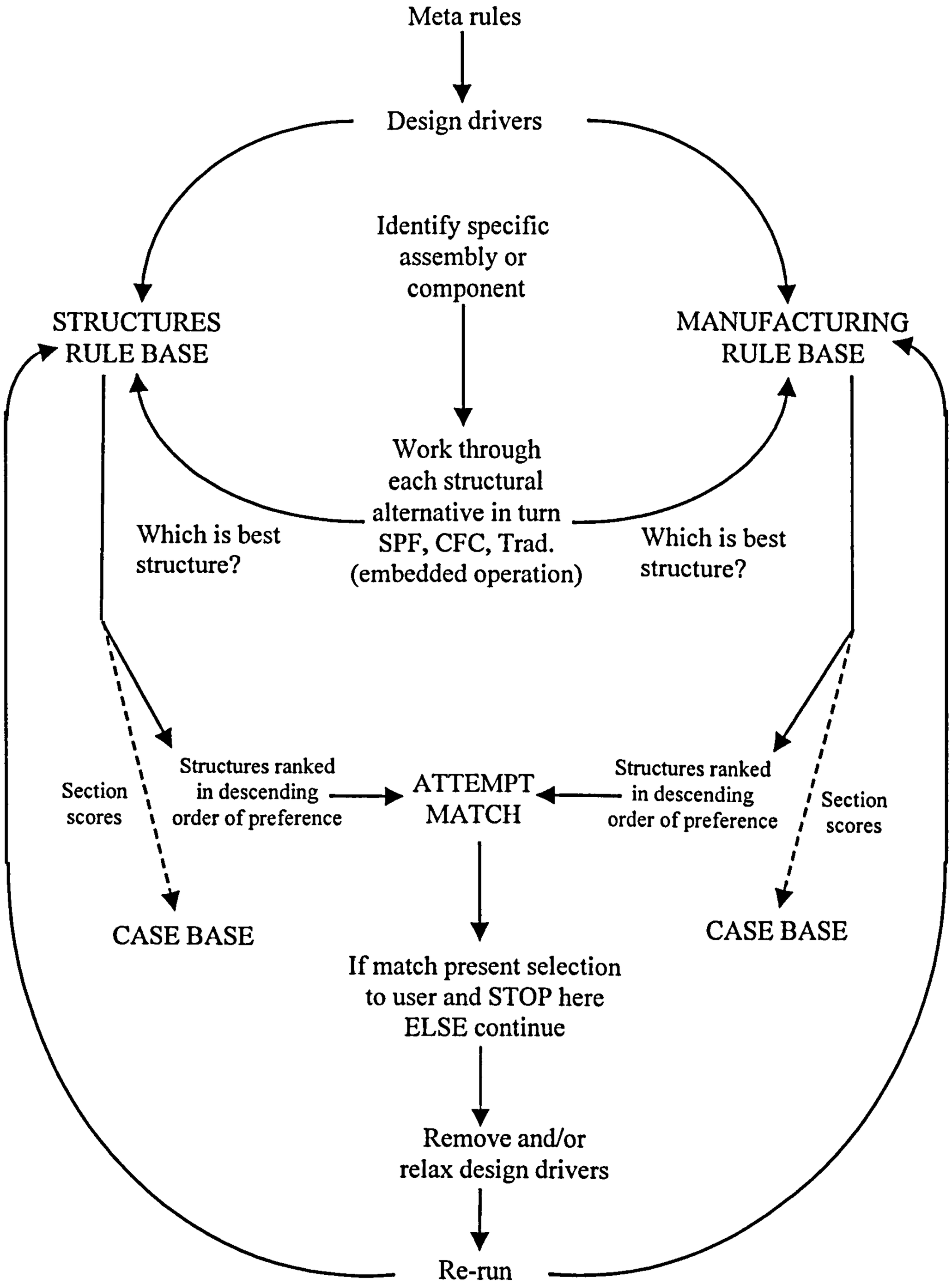



Figure 7.3, Manufacturing and Structures Rule Base Operation

**R&D Level**

1. The new design is to be developed relying solely on existing core competencies (this may or may not include an R&D element).
2. The development of the new design will aim to expand existing competencies.
3. The development of the new design will be looking at developing a totally new structure and thus gain new core competencies for the organisation.

The level of R&D selected impacts directly on the outcome of the KBS i.e., the best structure selected. Here ICES when it identifies the best structure will tell the user under what conditions the current level of R&D is appropriate. For example, suppose the user selects R&D level 1, where the design is intended to rely solely on existing core competencies. If the selected best structure happens to be SPF/DB titanium, ICES will suggest to the user that it will probably be necessary to increase the level of R&D.

### 7.2.4. The Manufacturing and Structures Rule Bases

Once the design drivers are identified and the importance ratings assigned by the meta rule base and the DDRT respectively, the ICES asks the user to identify the component of interest. This could be a specific individual component or conversely a main assembly. The system then proceeds to take each possible structural alternative i.e., SPF/DB titanium structures, CFC, traditional stressed skin, and present them to a structures rule base and a manufacturing rule base. Each rule base works through each structural alternative in turn. The design drivers defined by the meta rule base dictate the output of subsequent rules relating to each structural alternative under consideration. It should be noted that some rules are valid for both rule bases e.g., the selection of homogenous structure will impact on both rule bases. The structural alternatives are ranked in descending order of preference by each rule base. The ranked structural preferences from each rule base are then matched against each other. Where both the structures and manufacturing rule bases preferred choices are in agreement then this selection is presented to the user as the best structure. Figure 7.3 illustrates the basic operation of the manufacturing and structures rule bases and how the system attempts to derive a match between the two. If an agreed 'best structure' cannot be derived in this manner the system permits the user to relax the numerical weightings of design drivers acting on the manufacturing and structures rule bases. The system then re-runs and again attempts to achieve a match. Theoretically, by progressively relaxing the design driver weightings a match between the two rule bases will eventually be obtained.

#### 7.2.4.1. Operation of the Manufacturing and Structures Rule Base Rules

The output of rules operating in the manufacturing and structures rule bases impact on and are assigned to one or more of the following seven distinct sections:

1. Part Minimisation - To what degree can part reduction be achieved? Can homogenous sections be fabricated and how does this impact on structural and manufacturing requirements?
2. Load Carrying Capability - For each structure under consideration what are the load carrying characteristics? How do these load carrying characteristics affect manufacturing and structural performance?

3. Fastener Requirement - What is the requirement for fasteners and/or fastening medium? What fastening alternatives are available and how do these affect manufacturing and structural performance?
4. Fabrication - How can each structure under consideration be fabricated, and in what numbers? How does the mode of fabrication impact on structural capability? What are the manufacturing prerequisites of each mode of fabrication? What impact does the availability and cost of materials have on fabrication?
5. Environmental and Operational Characteristics - What are the environmental/operational characteristics of each section under consideration? How do these characteristics impact on manufacturing and structural requirements?
6. Geometric Considerations - What impact does the structures' geometry have on manufacturing capability and structural requirements?
7. Inter-connection of Assemblies and Subassemblies - The different modes of assembly and subassembly connection. How do these modes of connection impact on manufacturing and structural requirements?

The concept behind defining the above sections is that when rules fire in the manufacturing and/or structures rule bases each rule scores a weighting in one or more of these seven sections i.e., the firing of rules impacts on one or more of the sections. The weighted score for each rule is determined from a positive correlation between the rules residing in the rule bases and the design drivers. Listed below is the range of design driver/rule correlations and associated correlation factors used within this study.

| **Rule/Design Driver Correlation** | **Correlation Factor** |
|---|---|
| There is full correlation between the rule and the design driver. | 1 |
| The rule has a large correlation with the design driver | 0.8 |
| The rule correlation with the design driver is equal to any lack of correlation with the design driver. | 0.6 |
| The rule has little correlation with the design driver | 0.4 |
| There is no correlation between the design driver and the rule. | 0.2 |

The rule/design driver correlation factor is multiplied by the importance rating as derived by the DDRT. For both the manufacturing and structures rule bases the scores achieved in each of the sections are summed. The structure achieving the highest summation in each section is ranked highest and the structure achieving the highest summation in the most sections in a rule base, be it manufacturing or structures, is the preferred structure for that rule base.

As an example of the correlation between the rules residing in the rule bases and the design drivers, consider the following rule/design driver correlation for the manufacturing rule base where batch production is selected and carbon fibre composite (CFC) is the structure under consideration. It is important to appreciate here that batch production is itself a design driver and it is assumed that the start-up design driver is also deemed important to the designer. The weighted values of the design drivers are obtained from the DDRT. The output is assigned to one of the seven sections as described at the beginning of this section. For example, the score

obtained from the rule below is assigned to section 4 of the seven sections which relates to fabrication.

Section 4 = batch production design driver x 1 (where there is full correlation between the rule and the design driver)

What this relationship implies is that where batch production is required and batch production is the design driver, there is a very strong correlation between CFC and its ability to be manufactured in a batch production environment. As just indicated the numerical output of this rule is assigned to section 4.

Section 4 = start-up design driver x 1 (where there is full correlation between the rule and the design driver)

What this relationship implies is that where batch production is required and start-up costs is the design driver, there is a very strong correlation between CFC and the ability to spread or amortise the start up costs over the production run. The score obtained from this rule is again assigned to section 4 which relates to fabrication.

As a further example, consider the following rule/design driver correlation for the structures rule base where standard x-core and cellular core are the structures under consideration. It is required that the structure under design be able to carry out-of-plane loads. Here the correlation between the minimum weight design driver and the requirement for the structure under consideration to be able to carry out-of-plane loads is given for the standard x-core and cellular core structures.

*standard x-core*

Section 2 = minimum weight design driver x 0.4 (where the rule has little correlation with the design driver)

What this relationship indicates is that where it is desirable that the structure under design be able to carry out-of-plane loads there is little positive correlation between this requirement and the minimum weight design driver. What this relationship implies is that standard x-core's ability to carry out of plane loads is done so at the expense of the structure's mass. The score obtained from this rule is assigned to section 2 which relates to load carrying capability.

*cellular core*

Section 2 = minimum weight design driver x 1 (where there is full correlation between the rule and the design driver)

What this relationship indicates is that where it is desirable that the structure under design be able to carry out-of-plane loads there is a strong positive correlation between this requirement and the minimum weight design driver. What this relationship implies is that cellular core can carry out-of-plane loads with out any mass penalty. The score obtained from this rule is again assigned to section 2 which as indicated above relates to load carrying capability.

It is important to emphasise here that much of knowledge domain associated with the manufacturing and structures disciplines does not lend itself readily to the use of traditional production rules i.e., IF THEN statements. For example, when comparing cellular core structures to standard x-core structures, both are capable of carrying out-of-plane loads however cellular core with its ability to support ribs does so more efficiently than standard x-core. The rule/design driver correlation method presented attempts to account for this difference in structure performance.

The method described for deriving a match between the structures and manufacturing rule bases as outlined at the beginning of section 7.2.4 was implemented in the first ICES software prototype. This prototype is discussed in detail in Chapter 8. However, in the context of developing the ICES methodology it was proven to be flawed. This method of deriving a match between the manufacturing and structures rule bases failed because the system could not be guaranteed to achieve a match between the two rule bases (unless the design drivers were totally relaxed). It was conceivable that the system could perform many iterations and still fail to arrive at a best structure. It was therefore necessary to derive a strategy whereby a match between the structures and manufacturing rule bases could be guaranteed and thereby present a best structure to the user. The method developed to ensure a match between the two rule bases and which was subsequently implemented in the second ICES software prototype is now discussed in the following section.

#### 7.2.4.2. Ensuring a Match Between the Structures and Manufacturing Rule Bases

As indicated above, if the manufacturing and structures rule bases preferred choice is in agreement then this choice i.e., the 'best structure', is presented to the user. However, if the system is unable to achieve a match the user is asked to choose a preferred perspective i.e., to look at the problem from a manufacturing or structural perspective. Clearly, this will depend on what each rule base presents as the best structure. Having decided on the perspective to be taken the user now focuses on the opposing rule base i.e., if the structures perspective is the preferred perspective the user will focus attention on the manufacturing rule base and vice versa.

As the system has failed to deliver a match at this point the structure under consideration will not be the preferred structure in the opposing rule base. Thus, the procedure for deriving a match is as follows. In the opposing rule base those rules which have a design driver correlation factor which is less than '1' will have secondary rules assigned to them which dictate what must be done to achieve an improved correlation; section 7.2.4.1 provides an explanation of design driver correlation factors. It is important to note that each of the secondary rules provide an explanation of the consequences of carrying out the actions necessary to achieve the improved correlation between the design drivers and the secondary rules. Additionally, each secondary rule outputs an associated cost of action score or weighting. For all structures under consideration in the rule base, the scores obtained by multiplying the design driver correlation factors by the DDRT importance ratings are summed and the mean average of these scores is taken in the context of each design driver. Then for the structure under consideration each rule which has a correlation factor less than '1'

is identified and the appropriate secondary rule selected. The method for selecting the most appropriate rule is outlined in the following section. The secondary rule selected is then fired. The mean average, as indicated above, is again calculated. The improved score resulting from the firing of the secondary rule is now applied to the selected structure and a match between the manufacturing and structures rule bases is re-attempted. A match will be obtained if the structure in question has a score higher than that of other structures in the rule base. If a match is successful then the best structure as determined from the desired perspective is presented to the user. If a match between the structures and manufacturing rule bases is not obtained at this juncture then the next appropriate rule with a correlation factor less than '1' is selected. Figure 7.4 illustrates the changes made to the structures and manufacturing rule bases to ensure a match between the two rule bases preferred structure.

To avoid confusion, the reason for taking the mean average of the scores obtained by each structure in the context of the design drivers is to facilitate the saving of the solution where appropriate to a database. The saving of the previous solutions is commented upon in section 7.2.4.2.2.

#### 7.2.4.2.1 Selection of the Appropriate Rule to Change

Having identified the structure which it is desired to transform into the preferred structure for a particular rule base, it is necessary for the system to have a method for selecting the most appropriate rule (with a correlation factor less than '1') to be acted on first. The procedure used by ICES for selecting the first rule to be operated on is as follows. For the structure in question the highest scoring section is identified; the concept of sections was discussed in section 7.2.4.1. Having identified this section, each rule which contributes to the section score is examined to determine its percentage contribution. The rule percentage contribution scores are then compared against each other and the highest scoring rule is selected from those which have a correlation factor less than '1'. This rule will be the first to be considered for change. The next rule to be selected will be that which makes the next highest percentage contribution score, and so forth.

The justification for selecting rules for change in the above manner is twofold. Firstly, it is considered appropriate to select the section that impacts the most on the design i.e., any changes that are made will be relevant. Secondly, by selecting the highest scoring rule (with a correlation factor less than '1') changes that are made are relatively small but will be most effective in transforming the desired structure into the preferred one for a particular rule base.

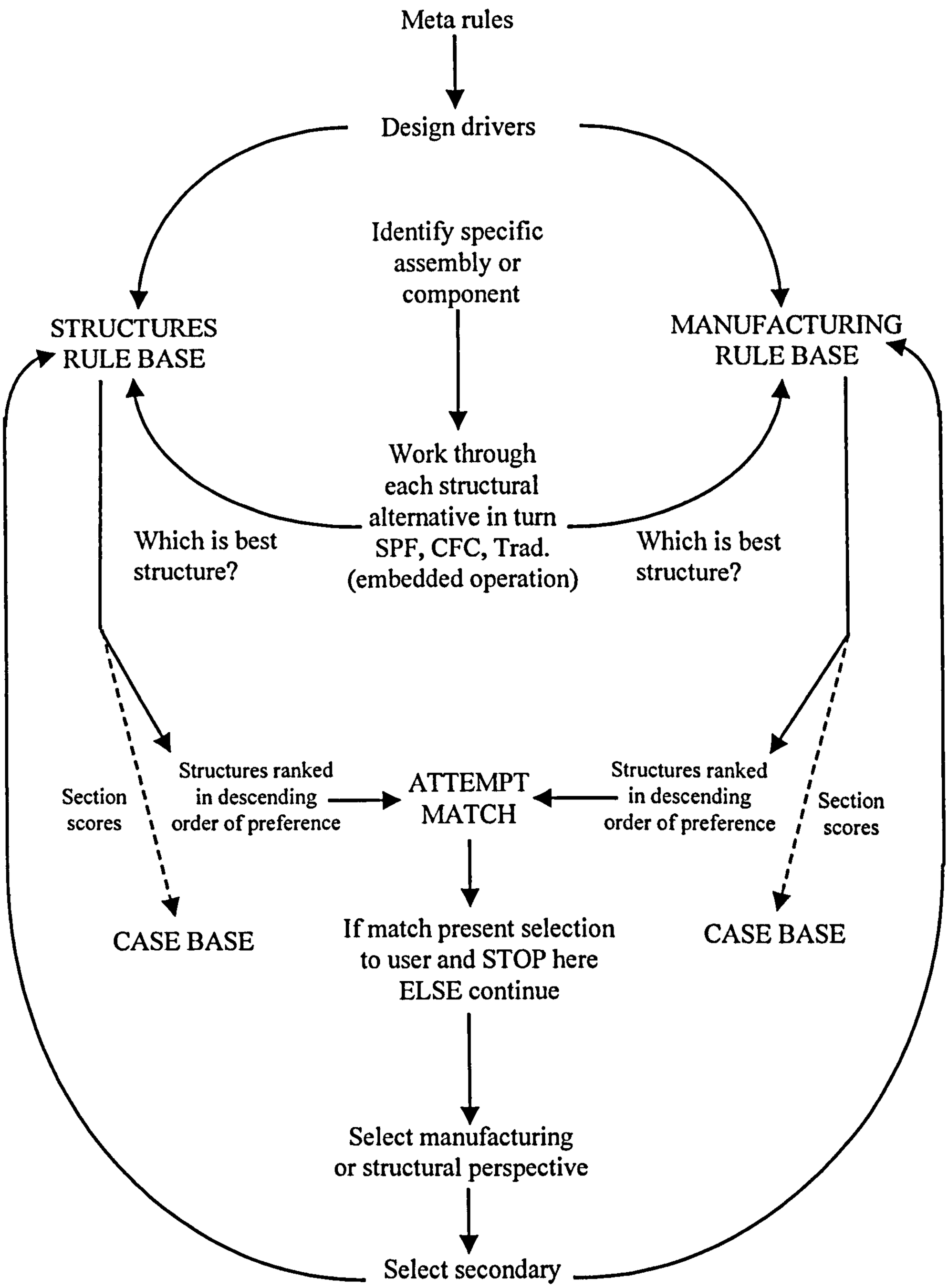



Figure 7.4, Method for Ensuring a Match Between the Structures and Manufacturing Rule Bases

**7.2.4.2.2. <u>Saving Previous Solutions</u>**

Having determined the best structure by embracing secondary rules, the ICES permits the solution to be saved to a database. This enables previous solutions to be used

when determining the best structure in the event that a similar situation is encountered.

#### 7.2.4.2.3. Cost Scores

As indicated above, cost weighting scores are assigned to the secondary rules that are provided to facilitate the derivation of an improved rule/design driver correlation factor. The cost weighting range is as follows:

- Expensive 1.00
- Costly 0.55
- Cheap 0.33

The range of cost weightings was deliberately kept to three in order to keep the process simple. The range was selected as indicated to achieve a significant numerical disparity between expensive, costly and cheap alternatives. As with the discussion concerning the numerical range of values assigned to order of magnitude relationships in section 7.2.2.1, there is no reason why management could not change the emphasis placed on the expensive, costly and cheap range. For instance, if management wanted to exclude all expensive alternatives as suggested by the secondary rules, then the expensive weighting score could be increased out of all proportion to the costly and cheap alternatives i.e., possibly by an order of magnitude. It is appreciated that costing is a discipline in its own right and there is certainly potential here for management research with respect to the costing decisions raised by the secondary rules and how the associated weights may be applied.

#### 7.2.4.2.4. Choosing Between Rule Base Operation or a Previously Derived Solution

The ICES has the capability to present the user with the best solution, be it via the rule base operation or directly from a previous stored solution residing in a database of previous solutions. The database contains the best previous solutions for each structure type i.e., SPF/DB titanium, CFC and traditional stressed skin. Stepping through the rule base the user is attempting to derive a better solution than the previous best solution stored in the database. If the user cycles through the rule base and fails to get a better solution than one stored in the database then this is obviously a waste of the user's time and computing resources. As such, ICES possesses the capability to arrive at the best structure as quickly as possible i.e., regardless of whether it resides in the database or the rule base. This is not a particularly important feature on the prototype documented in this thesis due to the small scale of the implementation. However, if the ICES methodology were to be developed into a 'fully fledged' system then solution retrieval time becomes a significant issue. It is conceivable that with a full implementation of ICES operating in a real world environment, cycling through the rule base will not be a trivial matter. Clearly, in such a scenario if a solution already exists it is desirable that it be accessed as quickly as possible.

In order to be able to discriminate efficiently between the rule base operation and a previous stored solution residing in the database the ICES uses a form of dynamic mapping. The dynamic mapping uses two numerical scoring procedures which are outlined in previous sections. The first weighting procedure relates to the seven section scores derived from the multiplication of the rule/design driver correlation factor by the importance rating (as derived by the DDRT), as discussed in section 7.2.2. The second weighting procedure relates to the cost of the solution obtained as indicated in sections 7.2.4.2 and 7.2.4.2.3. That is, cost weightings are assigned to those secondary rules that are provided to facilitate an improved rule/design driver correlation factor i.e., how much will any proposed changes cost?

The first stage of this process is to compare the stored solution score (as derived from the rule/design driver correlation factor and the DDRT importance rating) with that score required to make the structure in question the preferred structure. This comparison is necessary in order to ensure that the stored solution is from the outset valid i.e., if the stored solution is selected it will qualify as the preferred structure for the rule base. Assuming that the stored solution is capable of qualifying as the preferred structure, it is now necessary to determine whether a cheaper solution can be derived from the rule base. As each secondary rule is fired a comparison is made between the cost weighting scores of the new solution and the previously stored solution. If the cost of the new solution exceeds that of the stored solution prior to the structure in question becoming the preferred structure for that particular rule base, then the system will abandon the new solution in preference to the stored solution i.e., the stored solution will then be presented to the user.

If the rule base score for the structure in question is such that a match can be achieved between the structures and manufacturing rule bases without exceeding the old solution's cost weighting score then this new solution will overwrite the previous best solution stored in the database. Having achieved a match, the best structure is now presented to the user. An important feature to note is that this process is embedded within the system and not visible to the user. The reader is referred to section 4.1.2.2 of Chapter 4 which discusses the embedded operation of knowledge based systems.

## 7.3. Case Base

As indicated in the introduction to this chapter, the principle purpose of the case base is to permit the user who is in possession of design specifications to determine whether anything comparable to the current design requirement resides in the case base.

### 7.3.1. Structure of Case Base Knowledge

In the context of the ICES methodology, the knowledge associated with cases is structured using a frame-based approach. The frame-based approach for structuring domain knowledge was introduced in section 4.4 of Chapter 4. Figure 7.5 illustrates the structure of case knowledge using this technique. It can be seen from the figure that at the top level a frame is provided which accommodates the main assembly. This

frame has five slots which lead to the next level in the frame hierarchy and supports the main assembly engineering drawings, component description, performance information, information relating to sub-assemblies, and the output from the KBS.

The main assembly engineering drawing slot leads to a frame which provides access to engineering drawings which relate to the main assembly. The description slot leads to a frame which provides general information about the top level assembly. That is, the frame will discuss the purpose of the assembly and how it operates. For example, in the context of a foreplane assembly, the description slot will describe what the purpose of the foreplane is and what it facilitates in terms of aircraft performance. The performance slot leads to a frame which encapsulates the performance data i.e., primarily the flight envelope and environmental limitations. As can be seen from Figure 7.5, the sub-assembly slot leads to a frame which declares all the sub-assemblies which comprise the main assembly at the top level. This frame has five slots (assigned a letter for clarity i.e., a to e) which lead to the next and more detailed level in the frame hierarchy. At this next level the frame hierarchy supports frames associated with the

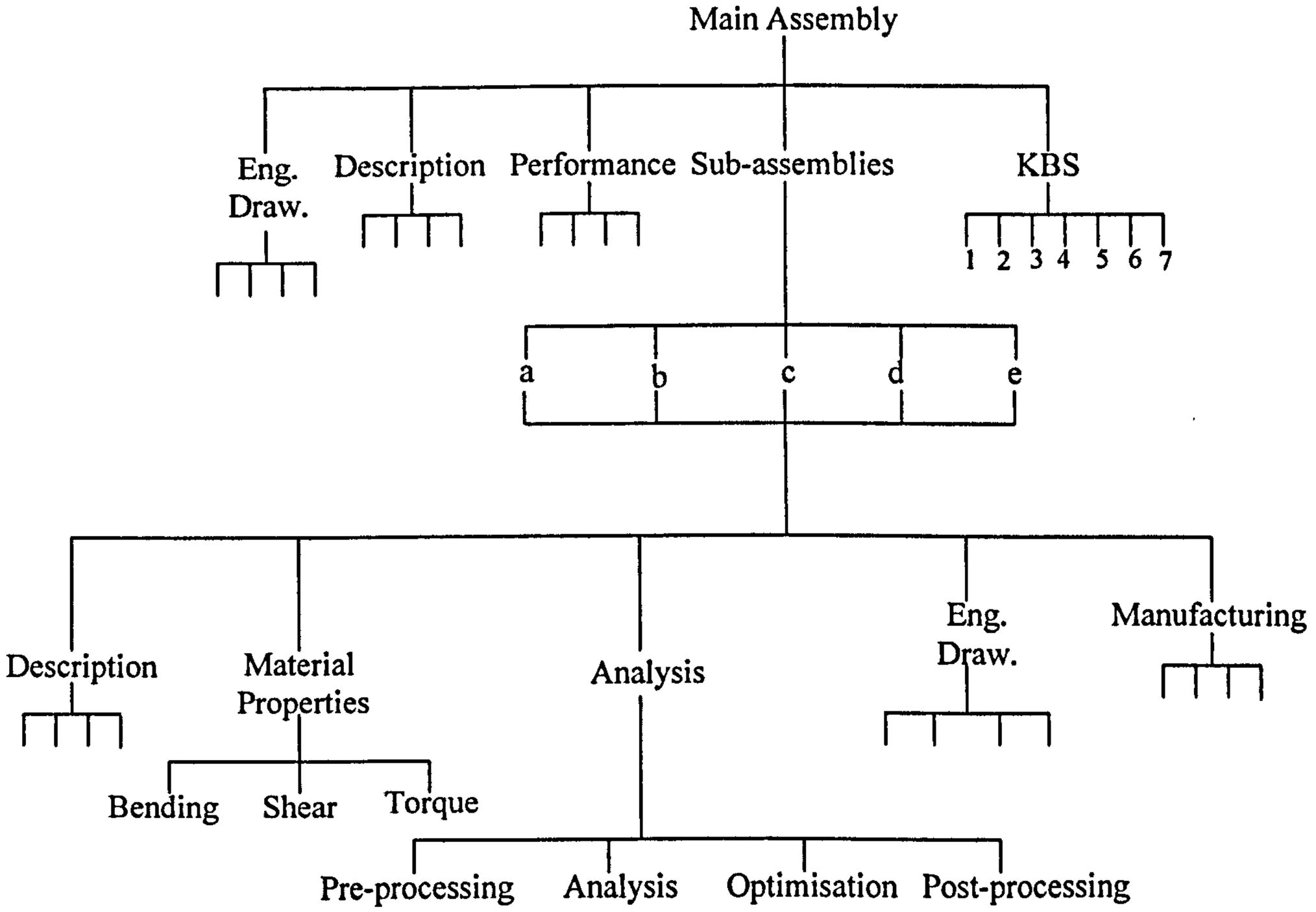



Figure 7.5, Case Knowledge Structured Using a Frame-based Approach

sub-assembly description, material properties, analysis, sub-assembly engineering drawings and manufacturing requirements; these frames themselves have further slots and associated frames. The KBS slot leads to a frame which supports the output from the KBS, specifically the seven section scores as defined in section 7.2.4. It should be noted that the relationship between the case base and the KBS is discussed in detail in section 7.4. It can be seen from Figure 7.5, that as one descends the frame hierarchy the level of detail provided by the design case progressively increases. There is no

limit to the number of frames or slots that a case may contain. However, there will be a limit to the amount of design information available and its usefulness with respect to presenting it in the manner described.

It is important for the reader to appreciate that to structure aircraft domain knowledge in the manner described i.e., using the frame-based approach, represents the preferred or ideal manner of structuring knowledge in this domain. In the 'real world' it is often the case that compromises have to be made. This, as will be apparent when discussing the ICES software and hardware development process and the ICES software implementation process in Chapters 8 and appendx D respectively, was certainly the case. As these discussions indicate, due to the limitations of the third party proprietary development software on which the ICES methodology was implemented it was not possible to utilise the frame-based approach described here. However, it should be emphasised that the frame-based hierarchical approach for structuring knowledge as described here and in Chapter 4 is the preferred method in the context of the ICES methodology.

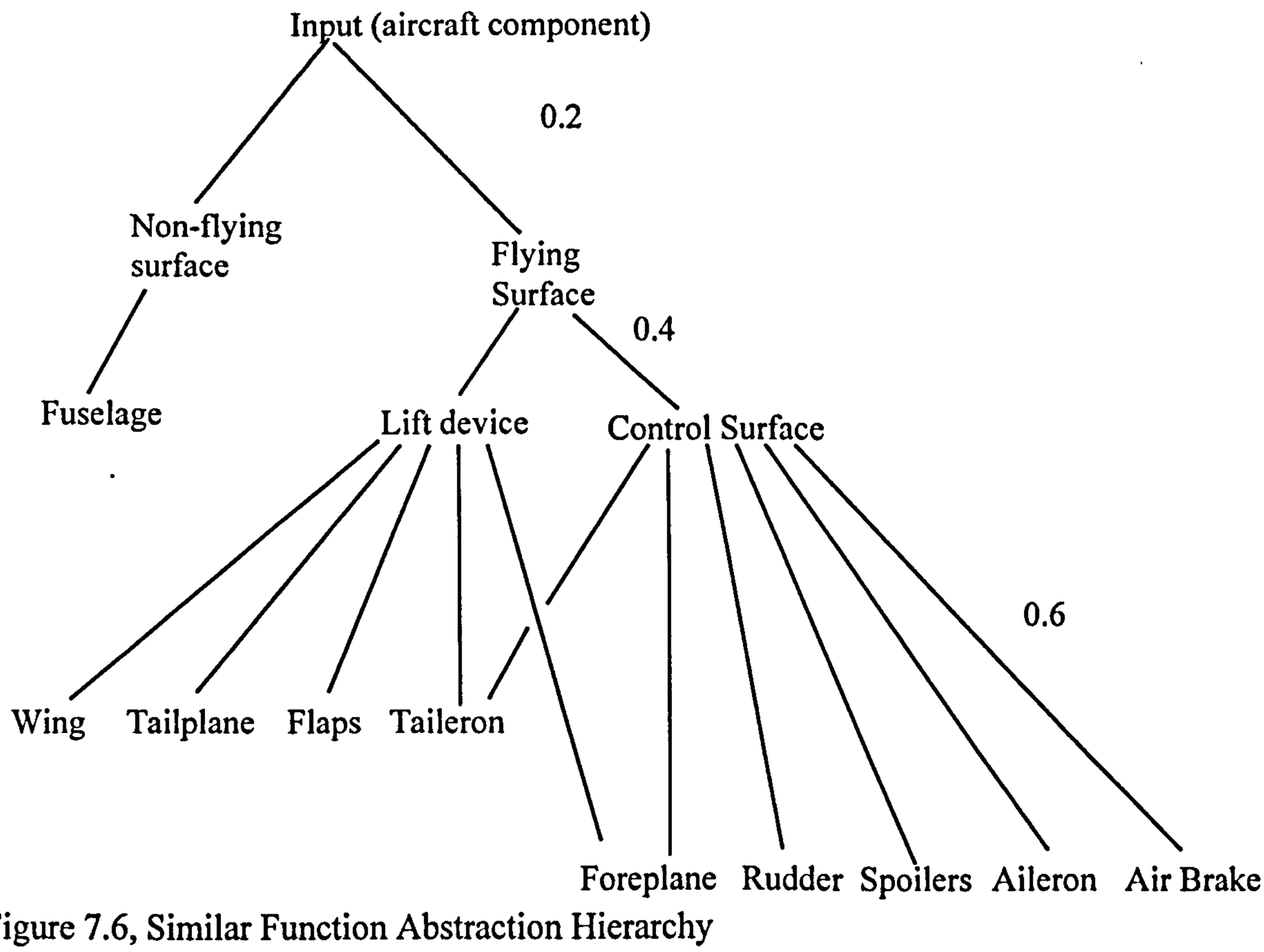



Figure 7.6, Similar Function Abstraction Hierarchy

### 7.3.2. Selection of the Best Case

The ICES case base requires that the user input the name of the structure of interest e.g., a foreplane. The system then determines the relative importance of the various structures residing in the case base. This is accomplished by using an abstraction hierarchy which indexes functionally similar structures i.e. a 'similar function' abstraction hierarchy. As can be seen from Figure 7.6, an abstraction hierarchy can be pictorially represented as an inverse tree-like structure. A different structural type is placed at each node on the tree. The tree starts with general functional information

about structures towards the root of the tree and then as one move towards the leaves of the tree the information becomes more specific.

Weightings are assigned to branches at each level in the tree. These become progressively higher the deeper one descends down the tree. The functional similarity between structures in the abstraction hierarchy is determined by summing the weighted values on branches of the tree leading to the structure in question. By comparing the scores attained by different structures the system determines their relative similarity with respect to function.

If the user is in a position to enter the structure type for the component to be manufactured (e.g., CFC, SPF/DB titanium or traditional stressed skin) then the system provides a 'similar structure' abstraction hierarchy. This abstraction hierarchy provides the means of identifying cases which possess the structure type in question and those representing a close match. The structure type abstraction hierarchy is identical with respect to mode of operation to the similar function abstraction hierarchy outlined above.

Finally, the user should enter all known additional specifications for the current design problem. Once these have been entered, the system compares them with the specifications of the stored cases residing in the case base. The system uses the *nearest neighbour* matching technique in order to compare the current design problem specifications with specifications of stored cases.[94] Nearest neighbour matching is used in conjunction with the abstraction hierarchy technique described above in order to determine the best matching case with respect to the current design problem.

It should be noted that due to the time constraints imposed on this study it was only practical to drive one structure through the ICES methodology in order to validate the concepts presented i.e., an aircraft foreplane. As such, it was not possible to implement abstraction hierarchies in the ICES software prototype presented in Chapter 9. However, it is envisaged that further implementations of the ICES prototype will see more than one aircraft structure supported. Thus, it will then be appropriate to implement the abstraction hierarchies as described above.

### 7.3.3. Case Selection Using a Jury Technique

Once the ICES case base has identified the best matching case it ranks all other cases in descending order of preference with respect to the abstraction hierarchies and other entered specifications. The system then presents the best matching case to the user.

It should be borne in mind that aircraft cases can potentially be huge. It is likely that the user, when entering specifications, will not have access to all the details that relate to the current design problem. Thus, it is possible that several aspects of the best matching case will be presented without justification i.e., aspects of the 'so-called' best matching case are presented on the grounds that it has more aspects in common with the specifications and abstraction hierarchies than any other case. Clearly, it is quite possible that those aspects of a case which make it a good or bad case may not bear any relation to the input specifications. The ICES methodology provides a very novel

solution to this problem. When the system presents the case that is the best match to the input specifications ICES requires that the unjustified aspects of the best case be defended with respect to the next 'best case' i.e., remembering cases are ranked in descending order of preference. The defence takes the form of the best case presenting the positive aspects of the case and presenting the negative aspects of alternative solutions. This is followed by the next 'best case' presenting it's best features and highlighting the failings of other cases.

If the best matching case is successful in defending itself against the next best case then the process will stop and the best matching case will remain as the best case. However, if unsuccessful, the best case may be rejected and the next best matching case becomes the best case. The new best case must now must defend itself against it's next best case. The process continues until a case in the position of being the 'best case' defends itself successfully.

The way that cases defend themselves is in some respects analogous to a court room scenario. Here the engineer or group of engineers acting as the jury decide whether an aspect of a case has been successfully defended or not. Because of this analogy with a court room scenario, this new and highly novel addition to case-based reasoning is called the 'jury technique'. Figure 7.7 provides a flow chart illustrating the underlying logic supporting the technique. The following discussion will now step through the operation of the jury technique as presented in the flow chart in order to clarify the operation of this novel addition to case-based reasoning (CBR).

As can be seen from the flow chart in Figure 7.7, the first action box presents the best matching case to the user. As indicated above, the best case is required to defend itself with respect to other cases residing in the case base. Once the best case defence has been presented it can be seen that a decision box is entered. This requires that decision be made with respect to whether the best case is the preferred case or not. At this juncture it is not possible to say whether the case is superior to other cases residing in the case base and therefore the No direction arrow is followed. This arrow leads to a further decision box which requires a decision to be made with respect to whether to reject the best case or not. Obviously, with out having accessed any other cases in the case base the No arrow is again followed. The No arrow leads to an action box causing the system to cycle round to the next best case. Following the direction of the arrows, the next best case presents it's defence. Again following the direction of the arrows, the next action box cycles the application back to the best case. The output from this action box leads back to the action box which presents the best case defence to the user.

After the best case defence is once again presented to the user, the direction arrow leads the user to the decision box 'is the best case the preferred case?'. If the next best case is judged to be superior to the current best case, the No direction arrow is taken. This leads to the decision box 'Do you wish to reject the best case?'. Clearly the response here is Yes and the Yes direction arrow is followed. The Yes direction arrow

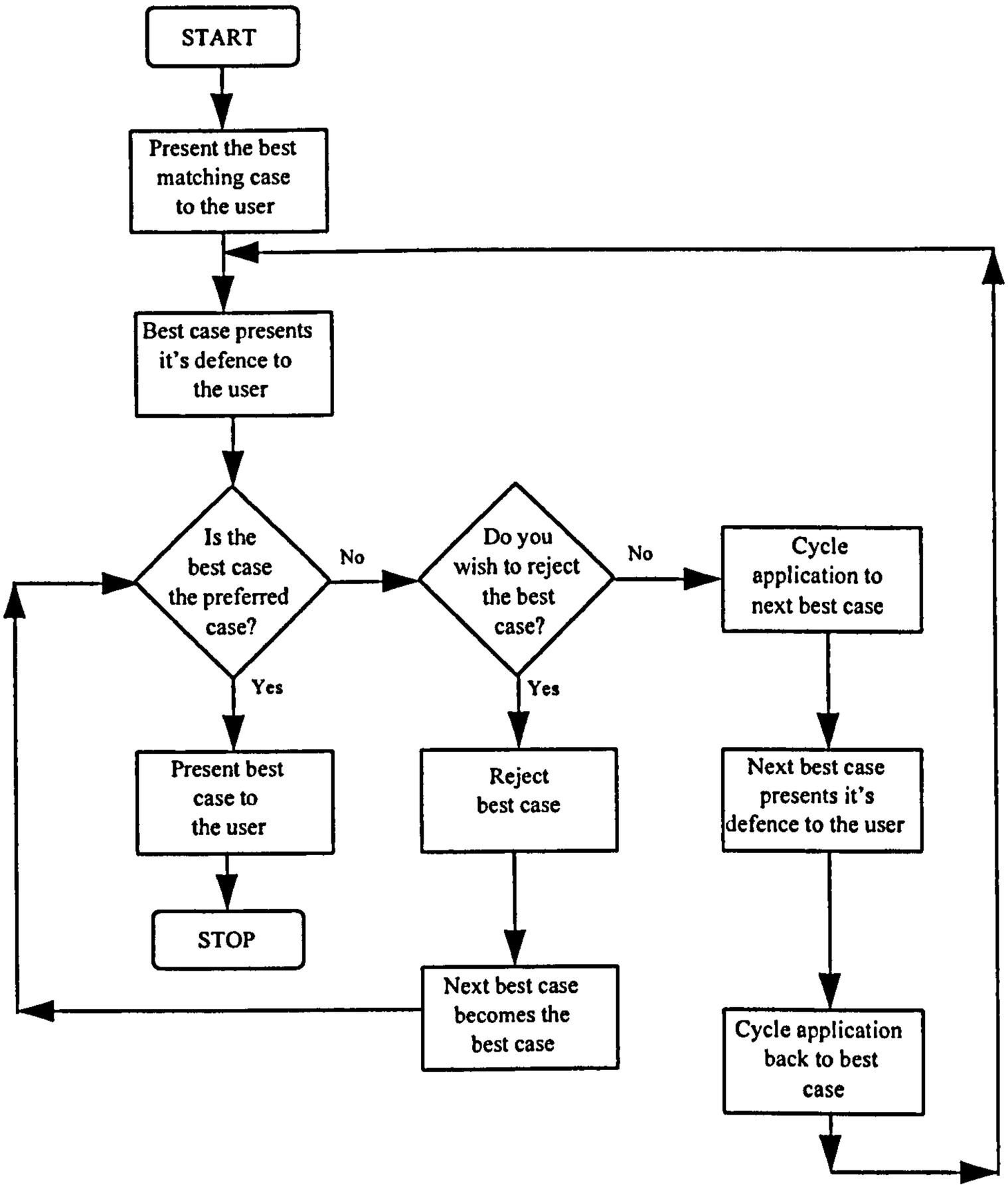



Figure 7.7, Flow Chart Illustrating the Underlying Logic of the Jury Technique

leads to two action boxes. The first causes the best case to be rejected and the second causes the next best case to be upgraded to the position of new best case. It can be seen from the flow chart that the direction arrow leading out of the second action box leads back to the decision box 'Is the best case the preferred case?'. At this juncture there is no other case to compare the new best case with so the response to this decision may cause the No direction arrow to be selected. However, if the user believes the new best case is indeed the best then this will cause the Yes direction arrow to be selected. This selection will lead to a single action box which causes the best case being presented to the user. The system has now reached the point of repetition and as such the logic of the jury technique has now been presented.

Clearly, in some instances there may be insufficient information for the system to argue the pros and cons of a particular aspect of a case. In this instance, each case, with the information available to it, presents to the user the risks of carrying on without further information to fill the technology gap(s). It was considered that there was potential here to assign numerical weights to the levels of risk involved with respect to case selection. This concept is discussed further in Chapter 11.

It is important for the reader to appreciate that the underlying rational for introducing the jury technique goes beyond purely the physical size of aircraft design cases. The technique takes on board other factors that may not be immediately apparent. The inherently long design cycle times in the aircraft industry make the application of traditional case-based reasoners inappropriate as they do not readily allow for incomplete cases or partial knowledge. In addition, the dearth of modern aircraft designs to place in a case base would up until the introduction of the jury technique have been the single most important influencing factor with respect to not selecting CBR for use in the military aircraft design domain. That is, up until this study, one of the 'corner stones' of CBR has been that there should be as many cases available as possible to make CBR a viable proposition. It is appreciated that there have been case-based reasoners developed with low numbers of cases before. However, in these situations it was always anticipated that the number of cases would increase significantly over time. This is not the situation with the jury technique as applied here. It is not anticipated that the numbers of military aircraft design cases available will increase significantly in the future. However, as shown in this study a very 'real' problem has been has been successfully accommodated within with this technique and made CBR a viable tool to be applied in the military aircraft industry.

Looking beyond this work, there is no reason the jury technique could not be applied to other design environments which suffer from similar problems as the aircraft industry in terms of long design cycle times and minimal cases being readily available. Candidate industries could be ship building or possibly the oil industry.

A further aspect of the jury technique is that it lends it self readily to multi-user input. That is, the 'jury' need not be a single designer but a design team. With the advance of the internet there is no reason why the jury could not reside in different companies and countries.

### 7.3.4. Case Adaptation

The ICES case base offers case adaptation in addition to purely just identifying the best case. In the situation where an aspect of a case has not been defended successfully the equivalent component in the next best case may be substituted. In this manner using *substitution techniques* a case may be adapted.[95] Adapted cases are then presented to the user as tentative design guides. It is important to note that adapted cases are segregated in the case base from real cases. This is because adapted cases are not as creditable as fully validated cases e.g., the European Fighter Aircraft (EFA) or the Experimental Aircraft Programme (EAP).

Where adapted cases are subsequently used they are presented to the user as 'best evidence'. As commented previously, adapted case data is not the same as validated data. However, some adapted cases will be closer to containing validated data than others i.e., with use the validity of case data increases. In order to discriminate between the validity of different adapted cases, each case is assigned a numeric confidence weighting. A case will increase its confidence weighting depending on how often it is used. Confidence weightings range between 0 and 1. A confidence weighting of 0 indicates completely invalidated data whilst a confidence weighting of

1 represents fully validated data comparable to a real case. As an adapted case can never truly become a real case any system for increasing the validity of an adapted case should not permit a case to automatically become validated. There needs to be a procedure whereby as an adapted case confidence weighting increases there becomes a point that the system 'flags' the user and suggests that the adapted case become validated. It is then up to the user to decide whether this is an appropriate course of action.

Identifying the confidence score at which an adapted case should be considered for validation is a difficult problem. There are no guidelines which can be referred to and as such, any method selected is very much a matter of 'trial and error'. It was therefore considered that the method developed should have a conservative bias, ensuring that a case is used many times before it approaches the validation score that will signal the system to flag the user to consider the case in question for validation. The method of assigning and incrementing confidence weightings used in this study is as follows.

A score of 0.1 is assigned to an adapted case each time it is used. This score is divided by 10 giving 0.01, 0.01 is then subtracted from 0.1 giving the adapted case an initial score of 0.09. When the adapted case is used again 0.1 is added to the previous score of 0.09 giving 0.19. This score is again divided by 10 giving 0.019 and this is subtracted from 0.19 giving a new adapted case score of 0.171. This process is repeated each time the adapted case is used. It should be noted that the adapted case score increases by an increasingly smaller percentage as it used. The justification for having a sliding scale of adapted case scores is that when a case is first used it indicates immediately that there is an element of validity about the case, otherwise it would never be used. However, the use of the sliding scale indicates the difference between a case being 'of use' and being considered validated. The difficulty of validating adapted design cases is indicated by the diminishing score. This approach is analogous with the concept of the 'learning curve'. When an adapted case is first retrieved there is much that can be used. However, when the case is used subsequently it may need to be further adapted to enhance its usefulness. This takes time and the return may not be as marked as before.

The confidence score at which the system will flag the user is very much a matter of trial and error and will vary according to the case in question i.e., possibly wing cases may be suitable for validation sooner than foreplane cases. In the context of this study it was considered that a confidence score of 0.7 may be an appropriate score at which the system should flag the user and recommend the case be considered for validation. Clearly, if this confidence score is set at too high a level i.e., a case becomes considered validated by engineers long before the 0.7 score is reached, then it will have to be revised.

There are a range of approaches to the concept of validating adapted design cases. There is no reason why the initial increase in adapted case scores could not be small but start to increase in size as the use-level crosses various thresholds. Further, it could be possible for an adapted case to flag the user each time it is used and query whether it should be considered for validation. It is clear from this discussion that the

concept of validating adapted aircraft design cases is an area which requires considerably more research. At the time of writing there is no evidence that one approach is superior to another. However, the approach used in this study is justifiable and possibly represents a useful starting point for further research.

### 7.4. Interaction Between the KBS and Case Base

When the ICES KBS has identified the best structure it is a requirement that the design case residing in the case base that represents the 'best match' to the structure under consideration i.e., as determined by the KBS, be presented to the user. The requirement to provide a link between the KBS and the case base in the manner described represents quite a novel problem. The reason being that in the majority of hybrid systems i.e., those systems that incorporate more than one knowledge source, in this instance rule bases and a case base, the knowledge sources tend to interact with one another, and together lead the user to the solution of the problem. However, with ICES, the case base and the KBS operate largely independently. Neither system component is reliant on the other to assist it with its solution. In order for the most suitable design case in the case base to be identified as a result of interaction between the KBS and the user there is a requirement to link the outcome of the *generic* rules residing in the KBS to the very *specific* design cases residing in the case base.

This link between the generic and the specific is achieved by utilising the scores obtained in the seven sections outlined in section 7.2.4.1. These seven discrete section scores are common to both the KBS and the case base. As already indicated, the concept behind defining the above sections is that when rules fire in the manufacturing and/or structures rule bases each rule scores a numerical value in one or more of these seven sections i.e., the firing of rules impacts on one or more of the sections. The score for each rule is determined from a correlation between the rules residing in the rule bases and the design drivers. Again, the operation of this technique is described in detail in section 7.2.4.1.

In the context of the case base, the section scores determined by the KBS for each structural alternative under consideration are assigned to the corresponding cases which utilise the structure in question. Each case has specific slots where each of the section scores are placed. As with the KBS manufacturing and structures rule bases the section scores are summed. The case that achieves the highest score is presented to the user as being the design case which represents the best match to the current design problem. Where more than one case utilises the structure in question it is possible for the user to transfer the first case details to the jury environment and query the case base further.

This chapter has provided a detailed outline of the ICES methodology, presenting reasoned arguments for the techniques employed. In the following chapter, the hardware and software issues that had to be addressed in order to facilitate the transfer of the ICES methodology into code are discussed. Following on from this discussion the ICES software prototype is presented in Chapter 9.

# Chapter 8

# ICES - Software and Hardware Development Process

This chapter discusses the principle software implementation issues that had to be addressed in order to enable the final version of ICES to be coded. This chapter traces the hardware and software selection and development process; the final software/hardware solution being Borland C++ Builder running on a Pentium PC.

## 8.1. The Software and Hardware Selection and Development Process

Before discussing the hardware and software selection and development process there is an important issue that it is necessary for the reader to first appreciate. The methodology discussed in this thesis has been evolved over the course of the research project i.e., a four year period. As the methodology evolved this naturally had an impact on the software selection and development process. At the early stages of the research project some of those software tools which it was considered suitable for the development of the prototype proved not to be so as the methodology evolved. In addition, it should be remembered that this research project was 'live' in as much that it had to embrace the on-going requirements of BAe MA&A. These requirements also had an impact throughout the duration of this study.

A further issue that may not be immediately apparent is that over the duration of the research project new software and hardware became available. For instance, at the outset of the research project the author only had access to 486 processor PCs. Whilst, towards the end of the research project high specification Pentium PCs with 32 bit architectures were available. In addition, a large range of affordable 'visual' programming software has become available over the past two years.

The remainder of this section now discusses the software and hardware available and its suitability with respect to building the prototype. Justification for the software and systems selected are given. As new software and hardware has become available its impact on the research project is discussed. The hardware and software available for this research project at the outset of the project is as listed below:

**Hardware**

- IBM RISC 600 250 workstations
- SUN workstations
- DECAlpha workstation 200
- DECworkstation 5000/200
- DECworkstation 3000
- Viglen 486 PCs

**Software**

- ICAD version 4.1. - knowledge-based engineering (KBE) system.
- CATIA version 4.1.2. - computer aided design (CAD) software package.
- I-DEAS - CAD software package
- Unigraphics - CAD software package
- C - programming language

- C++ - programming language
- FORTRAN - programming language
- Pascal - programming language
- BASIC - programming language

The selection of the appropriate software and hardware platforms on which to build the ICES prototype presented both technical and managerial issues. It was necessary to appreciate the repercussions of decisions made on both fronts. Clearly, there was a wide range of software and hardware platforms on which the ICES prototype could be developed. However, due to the various operating requirements and limitations at both BAe MA&A and Cranfield University certain items of hardware and software proved to be more suitable than others.

As a starting point, the requirements of BAe MA&A and the implications of these requirements were explored. BAe MA&A are committed to the use of CATIA, and as such during the early development stages of the prototype it seemed necessary that the prototype be able to interface with this CAD package. This requirement for the prototype to be able to interface with CATIA represented a significant factor with respect to selecting software for the prototype. Having recognised the potentially focal role that CATIA may have to play during the development of the prototype it was necessary to look at those items of software available that could be interfaced with CATIA. After examination of the relevant CATIA application architecture manuals it was revealed that CATIA supports interfacing with the FORTRAN and C programming languages. In addition, while not officially supporting the programming language C++ its successful interface with CATIA is acknowledged.[96] While appreciating the possibility of developing a prototype in C, C++ or FORTRAN and interfacing it with CATIA it was necessary to look at any other options that where available at this time.

The principle remaining option was the use of ICAD.[97] ICAD is the leading knowledge-based engineering system on the market and is available at Cranfield University. ICAD possesses the capability to interface directly with CATIA. It was recognised however, that the principle limitation of ICAD is that it does not possess any inferencing capability i.e., its search capability is very limited; it is not able to perform forward or backward chaining and is very much 'demand' driven. If ICAD where to be used, it would be necessary to develop an inferencing mechanism outside of ICAD. This inferencing mechanism would have to be capable of interfacing with both ICAD and CATIA. Considering that CATIA supports both C and FORTRAN it was decided to investigate the possibility of interfacing C with ICAD.

It was discovered that it is indeed possible to interface C with ICAD. However, for a successful interface with the C programming language it would be necessary to employ version 5 of ICAD. The version of ICAD supported at Cranfield University at the time of this investigation (December 1995) was version 4.1. Enquiries were made with respect to upgrading this version to version 5. Here a major 'stumbling block' was unearthed. The version of ICAD running at Cranfield University at this time ran on a SUN workstation whose operating system was Solarus 1. ICAD version 5 runs on Solarus 2. Unfortunately SUN at this time did not support Solarus 2 on this

particular workstation. With this information at hand it was apparent that it would not be possible to upgrade to version 5 of ICAD, and as such, there appeared to be no place for the system within the prototype development phase of this research project.

After exploring the possibilities of the software available and taking onboard the limitations imposed by both BAe, MA&A and Cranfield University it became apparent that the initial software selection that could be used for the prototype development was CATIA, C, C++ and FORTRAN.

As indicated in section 4.3 of Chapter 4, the object oriented methodology is ideal for the encapsulation of information from a wide range of disciplines such as those encountered during this research project. It was appreciated at an early stage during this study that C++ would have a prominent role to play in the development of the prototype. Of the three programming languages that it was initially considered for use with the prototype it was not envisaged that FORTRAN would have a predominant role to play other than that CATIA is written in FORTRAN i.e., a basic understanding of FORTRAN would all that would be necessary for the development of a successful interface.

Having completed this initial appraisal of the hardware and software available it was decided to attempt a C language interface to CATIA. It was decided to attempt the C interface rather than a C++ interface because IBM at the time of this investigation supported only the C interface to CATIA. However, as indicated above IBM did acknowledge that the C++ interface was possible. It was considered prudent to achieve the C interface first and then with the knowledge obtained from this exercise then attempt the C++ interface to CATIA.

In conjunction with this initial appreciation of the interfacing requirements of the ICES prototype a study was performed with respect to determine which programming language the prototype itself should be written in. Taking on-board the likely need to interface ICES with CATIA and acknowledging the potential benefits that the object-oriented methodology had to offer with respect to developing the ICES knowledge-based system (KBS) and the case base, it was considered that C++ represented the ideal choice of programming languages in which to code the first of two prototypes.

It should be noted that consideration was given to the purchase of a commercial case-based reasoning (CBR) system (e.g. ReMind, Kate). However, it was considered at this time that for the prototype development phase of this research project that the case base would be relatively small and not beyond the capabilities of the author to develop. In addition, it was also considered that as case base development progressed there would become a greater understanding of the requirements of a case base which would best fulfil BAe MA&A's needs. Thus, if appropriate, making any selection of a commercially available CBR system software more informed. Further, there was also concern that by purchasing an inappropriate CBR system, the prototype development would be unnecessarily restricted or limited due to the format in which information must be entered into the case base. It is worth pointing out that this restriction on data entry is a common problem with many expert system shells. This being one of the two reasons why an expert system shell was not considered appropriate for the prototype

development. The other reason being that it is not prudent to develop a methodology around a commercially based expert system shell which may cease to exist in the future or become incompatible with BAe MA&A's needs.

As with all projects, it is essential that those involved in the project have at a very earlier stage a clear appreciation of the aims of the project and what is required to satisfactorily meet those aims. In the context of this research project, the aim is to make a novel contribution to knowledge. While investigating the C programming interface to CATIA it was appreciated that the importance of being able to make a successful interface with this third party software package was negligible in the context of the overall project aims.

A basic C programming language interface to CATIA was achieved in early 1996. However, it was apparent that considerably more work would be necessary in order to achieve the level of interface that would be required if the first ICES prototype (when developed) was to operate successfully in the CATIA environment. It was decided that the first ICES prototype would focus solely on the functionality of the KBS and the case base; no attempt would be made at this stage (June 1996) to interface the prototype with CATIA.

The first ICES prototype was coded in C++, using Borland C++ version 3.1 and completed in November 1996. It was demonstrated to members of the Structures and Design departments at British Aerospace, Warton in December 1996. It is important to note that the purpose of a prototype is significantly different from a fully developed system, the two should not be confused. The under-lying aim of the prototype is to demonstrate the functionality of the proposed system and the potential that exists for a fully developed system. A prototype, while focusing on functionality does not attempt to present the user with the level of capability and user-friendliness that one might expect from a fully developed system. The principle aim of the prototype is to provide all interested parties with a focal point for constructive criticism and no more.

Before it was practical to present the first ICES prototype for evaluation it was necessary to take into account 'user expectation' during the evaluation process. This is because if the user finds the first prototype in a series of prototypes difficult to use this may cloud his or her view of the system and its potential. This in turn may make the possibilities for successful evaluations of future prototypes in the series harder as users have negative pre-conceived ideas about the technology being implemented. To overcome this potential problem it was essential that the prototype was delivered with appropriate supporting documentation. This documentation tells the user what he or she can expect in terms functionality from the prototype under evaluation. The documentation starts by introducing the prototype; what its purpose and aims are. This provides a general description of the system and how it operates. At a more detailed level the documentation discusses the level of user friendliness and error handling that can be expected. The documentation also discusses known operational limitations of the prototype. For example, the first ICES prototype did not have the capability to derive the 'best structure'. While this did not detract substantially from illustrating what was intended, it would have been undesirable for the user(s) to spend time attempting to derive a 'best structure' when at this juncture it was not possible. Finally,

the documentation should provide step-by-step examples to the various modes of operation of the prototype. This guide should start by telling the user how to start the prototype and thereafter lead the user through the system. At each step the guide should tell the user what controls to operate and the outcome of their subsequent operation. The documentation relating to the first ICES prototype is contained in appendix B.

Having indicated that a prototype should not be expected to have the same level of 'user-friendliness' and error handling as a fully developed system, it is worth here discussing briefly the level of capability present in the first ICES prototype. The first ICES prototype was not 'user-unfriendly'. However, the 'user-friendly' capability that was provided aimed solely at guiding the user through the operation of the prototype and no more. This first ICES prototype while operating in a Windows environment possessed no 'visual' capability; the prototype had a very 'MS-DOS feel' about it, possessing a scrollable menuing system. Unlike a fully developed system the prototype had no help facility. If the user encountered a run-time problem the documentation provided with the prototype provided sufficient information to get the user out of trouble.

To provide an acceptable level of error handling is probably one of the most time consuming aspects incurred during prototype development. It was not practical to attempt to provide totally comprehensive error handling. Instead, the error handling that was provided by the prototype focused on specific areas of the program's operation. Every attempt was made to ensure that the user could not enter contradictory information. The system informed the user if he or she attempted to do so e.g., if it was attempted to implement the design drivers for batch and one-off production at the same time the system would not accept this and an error would subsequently be reported to the user. During run-time the prototype was required to perform a large quantity of file handling operations i.e., entering data into a file, reading data from a file and counting data segments on a file. If any of the file handling failed during run-time the user was informed. As indicated above, it was not practical to provide fully comprehensive error handling and as such, known sources of error which were not covered in the prototype were documented in the accompanying prototype documentation.

In terms of functionality the first ICES prototype demonstrated a basic version of the KBS and case base as outlined in Chapter 7. As can be seen from the research project timetable in appendix D, a first prototype of ICES was developed in the second year of this research project (1996). The demonstration of the prototype at British Aerospace provided a useful forum to highlight the limitations of the functionality of the system presented. The issues raised during the demonstration fell into two categories; those which had an impact on the functionality presented and those which reflected on the underlying methodology on which the prototype was based.

With respect to the prototype functionality it was considered that the relatively poor quality of the user interface detracted from the system's operation. This limitation was acknowledged and countered by the fact that it was at this time the

intention to place the second ICES prototype within the CATIA environment and thus provide a user interface of an acceptable standard. A further issue relating to functionality was that some of the data output by the prototype's case base was not in an appropriate format i.e., information relating to design cases with carbon fibre composite (CFC) structures did not present all the information in a useful format. At this time it was considered that the application of an object oriented database may resolve this problem.

In terms of the underlying methodology three significant issues were raised during the demonstration and subsequent discussion. Firstly, the prototype used design driver weightings as discussed in Chapter 7. However, at this juncture these weightings were purely arbitrary and were not justified. It was therefore necessary to find a suitable method of deriving the weightings used. The weighting method eventually selected was the DDRT as described in Chapter 7.

The second issue related to the method of achieving a consensus between the structures and manufacturing rule bases preferred choice of structure. It was apparent that the initial method for achieving agreement between the two rule bases was flawed. The problem being that it was possible that a 'best structure' would not always be attained with the methodology at this level of development. The limitation of the methodology at this juncture is discussed in section 7.2.4.1 of Chapter 7. In addition, explanation is provided how the methodology evolved to overcome this problem.

The final issue which related to the under-lying methodology was the method of querying the case base. As aircraft cases have a tendency to be very large it is not appropriate to rely on a limited number of case specifications to provide a reflection of the quality of any particular case with respect to it being the 'best case'. This is because there may be other aspects of the case which are not considered when the specifications are initially entered; these may be critical with respect to deciding whether the case in question is really suitable. It was apparent that the traditional methods of case selection which are common to many CBR systems and were used within the first ICES prototype would have to be supported with additional new case selection methods. These methods would need to take a more detailed view of the suitability of cases with respect to them being considered for selection as the best case. This requirement for additional case selection methods lead to the development of the jury technique as outlined in Chapter 7.

Having evaluated the first ICES prototype it was not only necessary to overhaul the underlying prototype but also reappraise the software and hardware requirements. In the early stages of developing the methodology it was envisaged that the final ICES prototype would reside with the CATIA CAD environment. The reason for this was that it was considered likely that the system developed would in some manner drive or manipulate geometry residing in CATIA. Even before the completion of the first ICES prototype it became apparent that this would not be the case. It was now necessary to re-evaluate the role of CATIA within the research project. If geometry residing in CATIA was no longer to be affected by the operation of the ICES software what purpose and benefit could CATIA offer? In this context it was apparent that CATIA would only provide a 'visual' user interface. This interface whilst being

expensive i.e., every user of the ICES software would have to have a CATIA licence, would also limit the portability of the system; the ICES software could not be used outside of the CATIA environment. After re-appraising the potential role of CATIA within the development of the next ICES software prototype it was decided that there was no longer any justification for its inclusion.

Having decided not to use CATIA it was necessary to select an appropriate substitute package that would be able to provide a suitable visual front-end. It appeared on first inspection that a visual C++ PC based software package would satisfy the software requirements of the next ICES prototype. It is important to note that before selecting a visual C++ package it was necessary to gain an appreciation of the development and visualisation capability for PC based software in the Structures Unit at British Aerospace, Warton.

At the time of this investigation (March 1997), with respect to the development PC based application software, the Structures Unit were using Microsoft Visual C++. This PC based package was running on Windows NT. The intention being that any calculation code for applications developed in this package would be written such that it would be portable to any other environment i.e., UNIX. At the time of writing this thesis the transfer of code from the Visual C++ environment to UNIX was not possible.

In the context of visualisation it was possible to display a PC window generated by a PC-server in a UNIX environment. This was achieved by running a special version of Windows NT supplied by the company Tecktronics called "WinDD". It was envisaged that this situation would change when all development software work moved onto networked PCs running Windows NT. In the case of current networked PCs there is an emulator running on Windows NT which permits the user to connect to a UNIX environment and display UNIX windows (text and graphics) encased in a PC style window. The software that provides this capability is "Hummingbird Exceed" from the company Explorer UK. The Structures Unit, local area network (LAN) as of March 1997 is illustrated in Figure 8.1.

It is important to point out that the investigation into development and visualisation of PC based software in the Structures Unit took place in April 1997. Clearly, elements of the system will have advanced since this time. However, an important aspect of this investigation was that it revealed that it was possible to view PC based software and in particular Visual C++ applications on the majority of hardware platforms available at this time to the Structures Unit.

Having performed the investigation into the development and visualisation capability for PC based software in the Structures Unit at British Aerospace, Warton, it was decided to develop the second ICES prototype using visual C++. As the second prototype was to be developed at Cranfield University it was necessary to obtain a suitable hardware platform and appropriate software. In late 1996 Cranfield University had upgraded the majority of it's networked PCs to DAN Pentiums and had installed Borland C++ version 4.5. This software package being Borland's alternative to Microsoft's Visual C++. While it was appreciated that the Structures Unit at British

Aerospace were using Visual C++ and Cranfield University had Borland C++ version 4.5 installed it was not considered that this presented a problem. This is because the prototypes developed in this study are aimed at proving the viability of the artificial intelligence (AI) and computer based technology with respect to capturing and manipulating product knowledge and nothing more. It was never the intention throughout this research project that the final prototype would be used for anything other than demonstrating technology. With this in mind, it was not considered important which version of 'visual' C++ was used to develop the second ICES prototype.

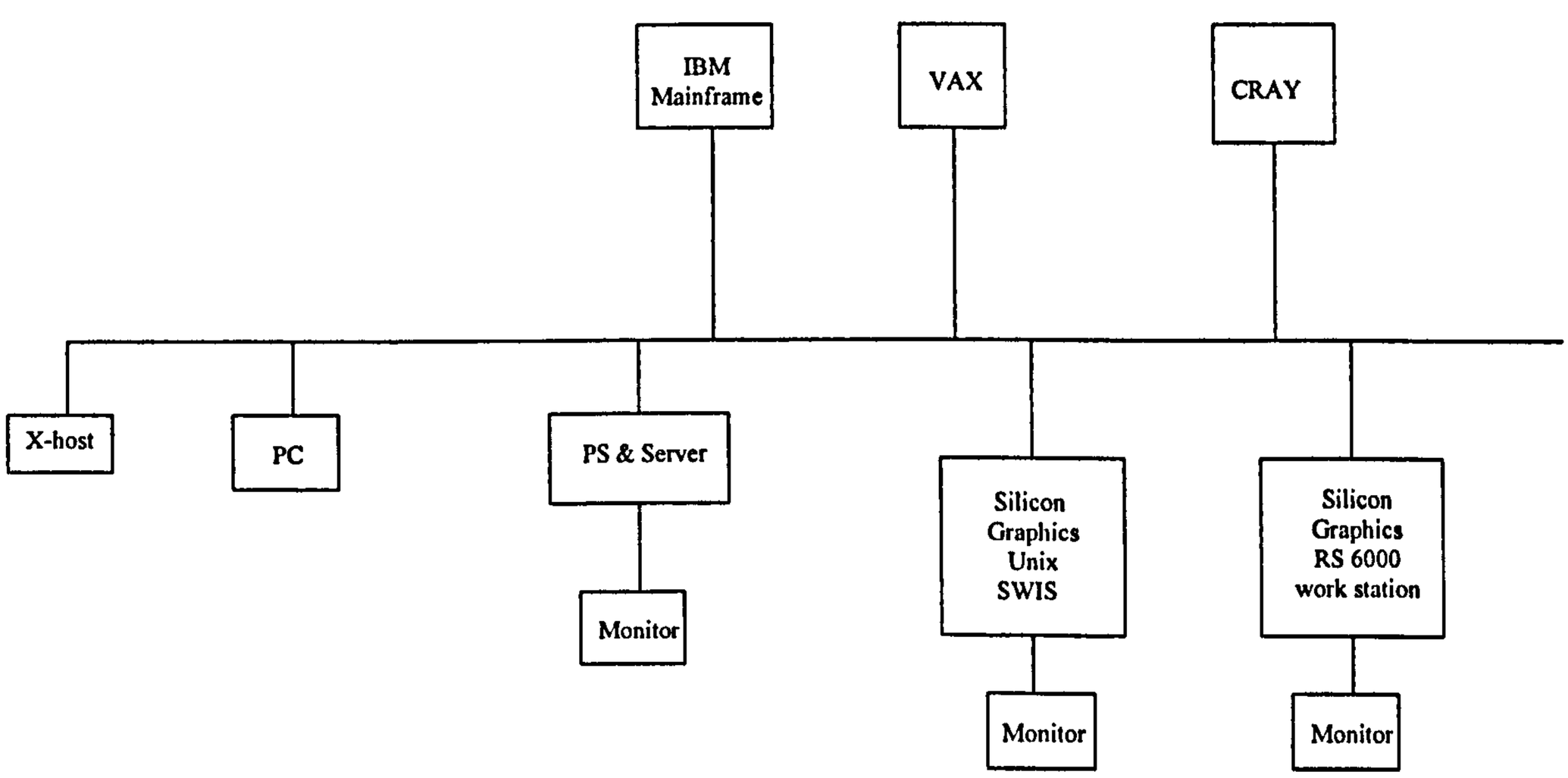



Figure 8.1, British Aerospace, MA&A - Structures Unit Network March 1997

Having taken on-board the issues raised at the demonstration of the first ICES prototype at British Aerospace, Warton in December 1996 the coding of the second ICES prototype started in June 1997. The prototype being developed on the Cranfield University PC network using Pentium PCs running Borland C++ version 4.5.

Shortly after starting to code the second ICES prototype it became apparent that the Cranfield University networked Pentium PCs were unsuitable platforms on which to develop the prototype. The networked PCs at this time were using the operating system Microsoft Windows 3.1 which itself runs on MS-DOS. Windows 3.1 is a 16 bit architecture. A significant limitation of a 16 bit architecture is that the maximum permitted segment size is 64 KB. What this means in programming terms is that no piece of source code is allowed by the MS-DOS to exceed this 64KB limit. If a segment of source code should attempt to exceed this limit a linker error is flagged, informing the user that the 64KB limit has been reached. With most C++ compilers it is possible overcome this problem by changing the type of memory model used to a flat memory model and as such there is no limit imposed on segment size. Unfortunately, Borland C++ version 4.5 does not offer the flat memory model option. The only possible way to code the second ICES prototype with Borland C++ version 4.5 would be to write the code in a series of code segments all of which must be

individually less than 64KB. The drawback with this option is that the problem being coded must lend itself to be broken down in such a manner. An attempt was made to code the second ICES prototype in this way but it soon became apparent at an early stage that the software being developed was becoming unwieldy. It was therefore decided to obtain a stand alone development platform running either Windows 95 or Windows NT. These operating systems provide a 32 bit development environment and as such there would be no restriction placed on the size of the code written i.e., it would be possible if desired to code the entire second ICES prototype as one single program. In addition to obtaining an alternative development platform it was also necessary to find an alternative to Borland C++ version 4.5 as this package cannot operate in a 32 bit environment.

An important point to note here is that BAe MA&A were kept abreast of the development of the second ICES prototype through regular reports and email contact. It is an important aspect of managing any significant project to document the progress of the project. The reports identify the progress being made and the problems (if any) that are encountered. In addition, it is good practice to identify the varying levels of risk involved with respect to taking a different course of action than the one initially intended. In the context of the problems presented by trying to code the second ICES prototype in a 16 bit architecture, there was no practical alternative to the course of action described. As such, documenting the risks to the research project was not appropriate in this instance.

By August 1997 a stand alone Dell Pentium computer running Windows 95 was obtained. An investigation into software alternatives to Borland C++ version 4.5 which were capable of running on Windows 95 identified Borland C++ version 5.0 as the most appropriate solution. In terms of functionality Borland C++ version 5.0 appeared to offer significant benefits in comparison to Borland C++ version 4.5. Borland C++ version 5 comes with a visual database tool capability; it would be possible to provide a software link to a proprietary database package i.e., Paradox or DBase. It should be noted that some form of database capability would be required to facilitate the case base operation in the second ICES prototype. The first prototype described above only possessed a token database capability i.e., just sufficient to provide the desired functionality and was coded in its entirety by the Author. It was intended that the second ICES prototype would provide a fully fledged database capability. Section D.2 of appendix D discusses the various aspects of the database selection.

The principle reason for selecting Borland C++ version 5.0 was that in terms of operation it was nearly identical to Borland C++ version 4.5; implying there would be a shallow learning curve for the Author to climb to enable the prototype to be continued. In addition, with minor changes, it was possible to take existing code written in Borland C++ version 4.5 and compile it in Borland C++ version 5.0.

From August 1997 through to the end of December 1997 the coding focused on the KBS component of the second ICES prototype. At the beginning of 1998 it was decided that the KBS was sufficiently developed to be able to switch attention to the case base. As the case base capability of the prototype would be reliant on the being

able to utilise Borland C++ version 5's database tools it was appropriate to experiment with them in order to gain an appreciation of how they should be used. The database tools were supposed to provide a software link between the application being developed with Borland C++ version 5 and a propriority database; Paradox version 7 being the database used in this case, see section D.2.5 of appendix D. It is important to point out that visual database tools only provide the links to an existing database and do not in themselves facilitate the construction of a database.

Through experimentation is was revealed that the database tools provided with Borland C++ version 5 did not function properly and could not be used within the prototype. The first indication that there was a problem with the database tools was that very few of the on-board examples could be compiled and run. Those examples that did run provided no real indication of how the database tools operated. At this juncture it was decided to seek advice. From experience it was known that the internet would provide the best source of advice. After placing queries with known Borland C++ user-groups it was revealed that Borland C++ version 5 database tools were generally considered unsuitable for database application development. The principle reason for the database tools failure was that they were full of 'bugs' and as such were generally not used by those software developers trying to code serious database applications. The preferred Borland C++ software package used by database developers was Borland C++ Builder.

The problems with Borland C++ version 5 discussed in the previous paragraph permitted two possible courses of action to be followed. Firstly, it would be possible to continue the development of the prototype with Borland C++ version 5 and attempt to overcome the software bugs as and when they presented themselves during the continuing development of the prototype. The second course of action would be to move away from Borland C++ version 5 and use Borland C++ Builder instead. The determining factor here was time i.e., as can be seen from the research project plan time table enclosed in appendix E, it was intended to complete the second ICES prototype by the end of December 1997; by the end of January there was still the case base component of the prototype still to code. Therefore, whatever choice was made, it was necessary to be certain, as far as possible, that the software solution selected would eventually permit the prototype to be satisfactorily completed. There were pros and cons with respect to both options. If it was decided to stay with Borland C++ version 5 the majority of the KBS component of the prototype was near completion; if the visual database tools bugs that were present in the software could be overcome the potential time to completion of the prototype could be quite short. However, it was possible that the prototype development could become 'bogged-down' in a futile effort to solve un-solveable Borland C++ version 5 source code bugs.

If it was decided to change the software platform to Borland C++ Builder it would be necessary to transfer all the existing code developed in Borland C++ version 5 across to Borland C++ Builder. It was considered conservatively, that this task would take at least 4 weeks to complete. However, on the positive side, after discussion with C++ Builder users it was apparent that the package's database interface was vastly superior to Borland C++ version 5 and was possibly one of the best tools of it's type on the market. It is worth pointing out that C++ Builder is a

rapid application development (RAD) tool and facilitates the quick creation of prototypes. This capability is also to be found in Microsoft's Visual Basic. However, unlike Visual Basic Borland's C++ Builder is also a fully fledged application development tool.

After taking on-board the development issues relating to both software packages it was decided to move the ICES prototype development away from Borland C++ version 5 and use Borland C++ Builder. The principle reason for choosing C++ Builder was that by using this package it was more certain that the prototype could be completed. Clearly, there was learning curve to climb with respect to using C++ Builder. The first part of this learning process was to investigate the operation of the package's database interface capability. The Author was pleasantly pleased to discover that the database links provided by C++ Builder were as good as described. A point worth noting here is that C++ Builder bears some similarity with respect to the development platform layout to that of Borland C++ version 5. This made the C++ Builder learning curve relatively easy to climb.

In terms of the software/hardware selection and development process, Borland C++ Builder running on a 166MHz Pentium PC was the final solution. The prototype was satisfactorily completed in June 1998 and subsequently demonstrated at British Aerospace, Warton in July 1998. Appendix D outlines the salient features of Borland C++ Builder and highlights the limitations of Borland C++ Builder in the context of developing the ICES prototype. In addition to discussing the principle features of C++ Builder, Appendix D makes a comparison between the relational and object oriented database methodologies. This is an important issue as case-based reasoning (CBR) plays a significant role within ICES and as such, it was necessary to consider the various database options that were available when developing the system. Appendix D focuses on the limitations that C++ Builder and the two database methodologies imposed on the eventual database selection. As indicated in Appendix D, Paradox version 7 was the database software packaged selected to use for the development of the case base component of the ICES prototype.

Having discussed the software and hardware selection process, Chapter 9 now leads on to describe how the ICES methodology was implemented in Borland C++ Builder.

# Chapter 9

# ICES Software Implementation

This chapter describes in detail the software implementation of the ICES methodology as documented in Chapter 7. The methodology is encapsulated within the second ICES software prototype which was coded using Borland's C++ Builder. The purpose of the prototype is to validate the concepts presented in the ICES methodology. As indicated in section 5.2 of Chapter 5, the process of validation or testing of the prototype requires that an appropriate structure be selected and driven through the system. In the context of this study it was considered that an aircraft foreplane represented such a structure.

The chapter starts by introducing the ICES prototype; this provides an overview of the software. The chapter then proceeds to provide a comprehensive and logical description of the ICES prototype operation. This explanation is supported throughout with illustrations of the range of output that the ICES prototype provides. Throughout the description of the ICES prototype appropriate reference is made to the ICES methodology as presented in Chapter 7. It is important at this juncture for the reader to appreciate that the underlying code that supports the ICES prototype is extensive. In addition, as it is the purpose of the this chapter to provide an understanding of the ICES prototype operation, it is deemed undesirable to present extensive detailed explanations of how the underlying code supporting the ICES prototype functions. However, it is appreciated that in the context of providing the reader with more detailed information with respect to the operation of the prototype there is a need to provide a detailed explanation of how the underlying code supports the prototype's functionality. In addition, it is conceivable in the future that it may be desirable to further develop the prototype as presented in this thesis. For a more detailed explanation of the ICES prototype supporting code the reader is direct to appendix C. Here a description of how the different component parts of the ICES prototype interact is provided and in particular focus is placed on how information is passed between components. It should be noted in the context of the code examples presented in appendix C, it is assumed that the reader has a basic understand of the C and/or the C++ programming languages. The final section of this chapter provides an example run of the ICES prototype and presents the results output by the system.

## 9.1. Overview of ICES Software

The ICES methodology as described in Chapter 7 identifies knowledge-based system (KBS) and case base reasoning (CBR) as appropriate artificial intelligence (AI) technologies to facilitate the capture of product knowledge in the conceptual design arena. These two AI technologies are implemented within the ICES prototype described in this chapter. As indicated above the prototype was developed using C++ Builder. This is a rapid application development (RAD) tool which facilitates the development of C++ applications in a 'visual' environment. In terms of user operation this means that the system's inputs and output are based around a Windows style environment. That is, the user input and system output is accepted and presented through the use of pull-down menus, dialog boxes, edit boxes and memo boxes etc.

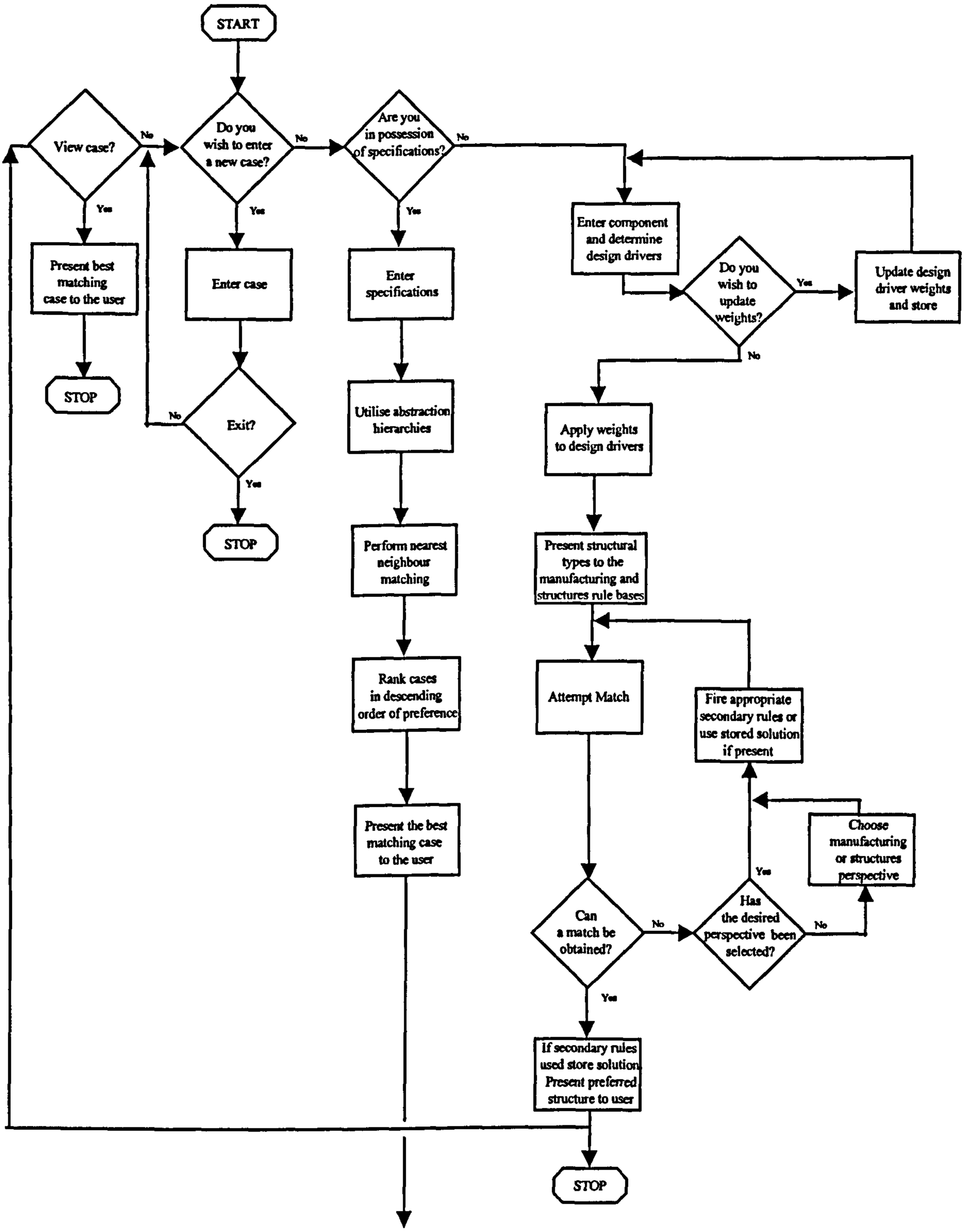
This Windows style user interface has significant advantages compared to user interfaces based on procedural programming techniques i.e., having an MS-DOS scrolling screen style interface. Where an application such as the ICES prototype, as presented here, offers a range of capability the user may interface only with that part of the application he or she is interested in. In addition, the Windows style user interface readily permits the user to re-trace his or her steps and alter or cancel previously made inputs. It is then possible, in many instances, for the user to proceed from the position in the application prior to making such changes and carry on as before. Further, the Windows interface greatly facilitates the placement of 'help' utilities throughout the application. These may be user selected or operate relatively automatically. Both types of help facility are implemented within the ICES prototype and discussed within this chapter.

As indicated above the ICES prototype supports two AI technologies, a KBS and CBR. These two technologies residing within the prototype provide two distinct modes of operation. The KBS supports three integral rule bases i.e., a meta rule base, a structures rule base and manufacturing rule base. Together, these rule bases enable the user to identify the 'best structure' from a combined manufacturing and structures perspective. As will be illustrated in subsequent sections of this chapter the KBS is generic in nature i.e., it does not focus on one particular aircraft structure. The concept underlying the operation of the KBS is that the user will have already identified the component it is desired to design but requires guidance with respect to which of the structures currently available to utilise e.g., traditional stressed skin, carbon fibre composite (CFC) or possible one of a range of super-plastic formed defusion bonded (SPF/DB) titanium structures. On the basis of user inputs the ICES prototype will recommend a preferred or best structure. Subsequent sections within this chapter will explain in detail how the ICES prototype implements the KBS in the context of arriving at this best structure.

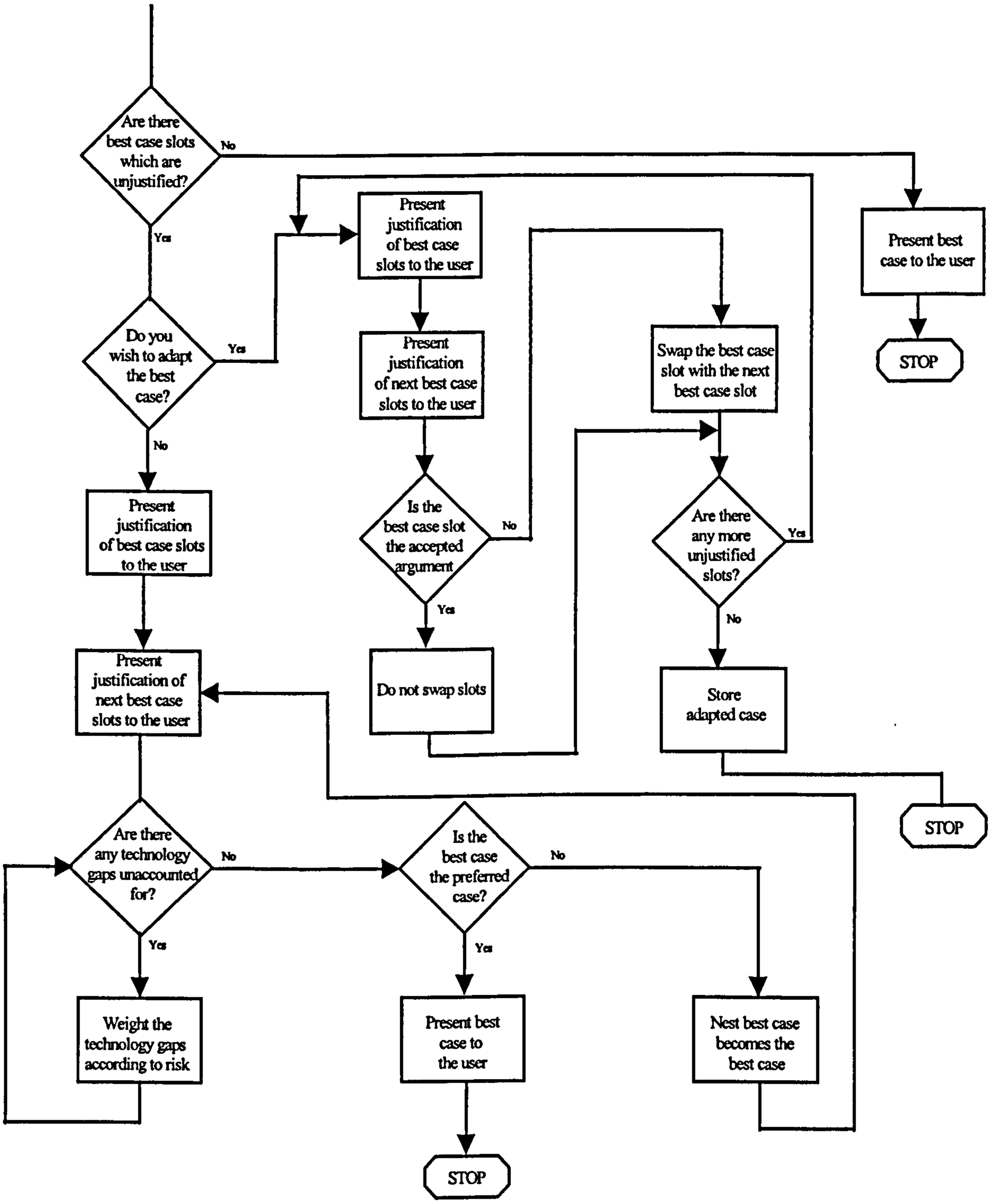
Turning attention now to the operation of the case base. As with the KBS, the CBR mode of operation is generic, i.e., the implementation of the methodology is not specific to any one particular aircraft structure. However, as indicated in the first paragraph of this chapter it was necessary to drive a specific aircraft structure through the ICES prototype in order to validate it. The structure selected was an aircraft foreplane and residing in the case base are a series of aircraft foreplane cases. As is discussed in appendix C, the ICES prototype case base has the capability to be expanded to support a range of different aircraft structures.

The user interacts with the case base in a different manner to the KBS in as much that the user not only knows what aircraft structure he or she wishes to design but also has access to specifications. Once the user has entered these specifications the case base presents to the user the best matching design case that resides in the case base. As indicated in section 7.3.3 of Chapter 7 this is not the end of the story. The case base operates a novel jury type system whereby the best case must defend itself against the next best case. Only when the best case has successfully defended itself is it truly the best case. This jury system as implemented within the ICES prototype is described in detail in section 9.2.3.2.2 of this chapter.

Figure 9.1, Structure of ICES Prototype




Continued overleaf




As indicated in section 7.4 of Chapter 7, there is a relationship between the KBS and the case base. Once the KBS has identified a best structure it is possible to go to the case base and identify the design case residing in the case base that represents the 'best match' to the structure under consideration i.e., the case(s) that possess the best structure. This aspect of the ICES prototype is discussed in section 9.2.3.2.1. Having provided the reader with a brief overview of the ICES prototype operation the following section now describes the underlying structure of the software supporting this application. The flow chart illustrated in Figure 9.1 clearly presents the structure of the ICES prototype developed within the C++ Builder environment.

## 9.2. ICES Software Operation and Description

While the previous section provided an overview of the ICES prototype this section now explains the operation of the prototype. The section takes the reader through the prototype in a logical step-by-step manner. The section begins by discussing the prototype's 'front-end' i.e., what the user sees on start-up and in particular the operation of the main menu. The section then leads on to discuss the salient features of the KBS, focusing on the meta rule base, the design driver ranking technique (DDRT), and the manufacturing and structures rule bases. The section concludes by exploring the case base operation. Here the discussion focuses on how a new design case is entered into the case base; how the case base is queried, in particular, with respect to the application of the jury technique; and finally an explanation is provided of how design cases may be adapted.

### 9.2.1. ICES Software - 'Front-end'

The 'front-end' of the ICES prototype is illustrated in Figure 9.2. This presents the main menu to the user. There are presented six menu options; File, KBS, Component, Options, Case Base and Help. These menu options provide a range of pull-down sub-menus. The operation of the main menu and associated sub-menus is now described below. The code that supports the operation of the front-end is provided in section C.2 of appendix C. It should be noted that main menu options descriptions are presented in a manner which facilitates the overall understanding of the ICES prototype rather than in the order that they appear in Figure 9.2.

#### 9.2.1.1. File

The File menu supports the pull-down menu option Exit.

##### 9.2.1.1.1. File Menu Options - Exit

The Exit menu option in the File menu permits the user to close down the ICES prototype. The code that supports the Exit menu option is explained in section C.2.1 of appendix C.

#### 9.2.1.2. Component

On start-up if the user wishes to initiate the KBS capability of ICES prototype application this is the menu option that should first be selected.

##### 9.2.1.2.1. Component Menu Options - Enter Component

On selecting the Enter Component menu option from the Component menu, the system presents to the user the Structure Identification dialog box illustrated in Figure

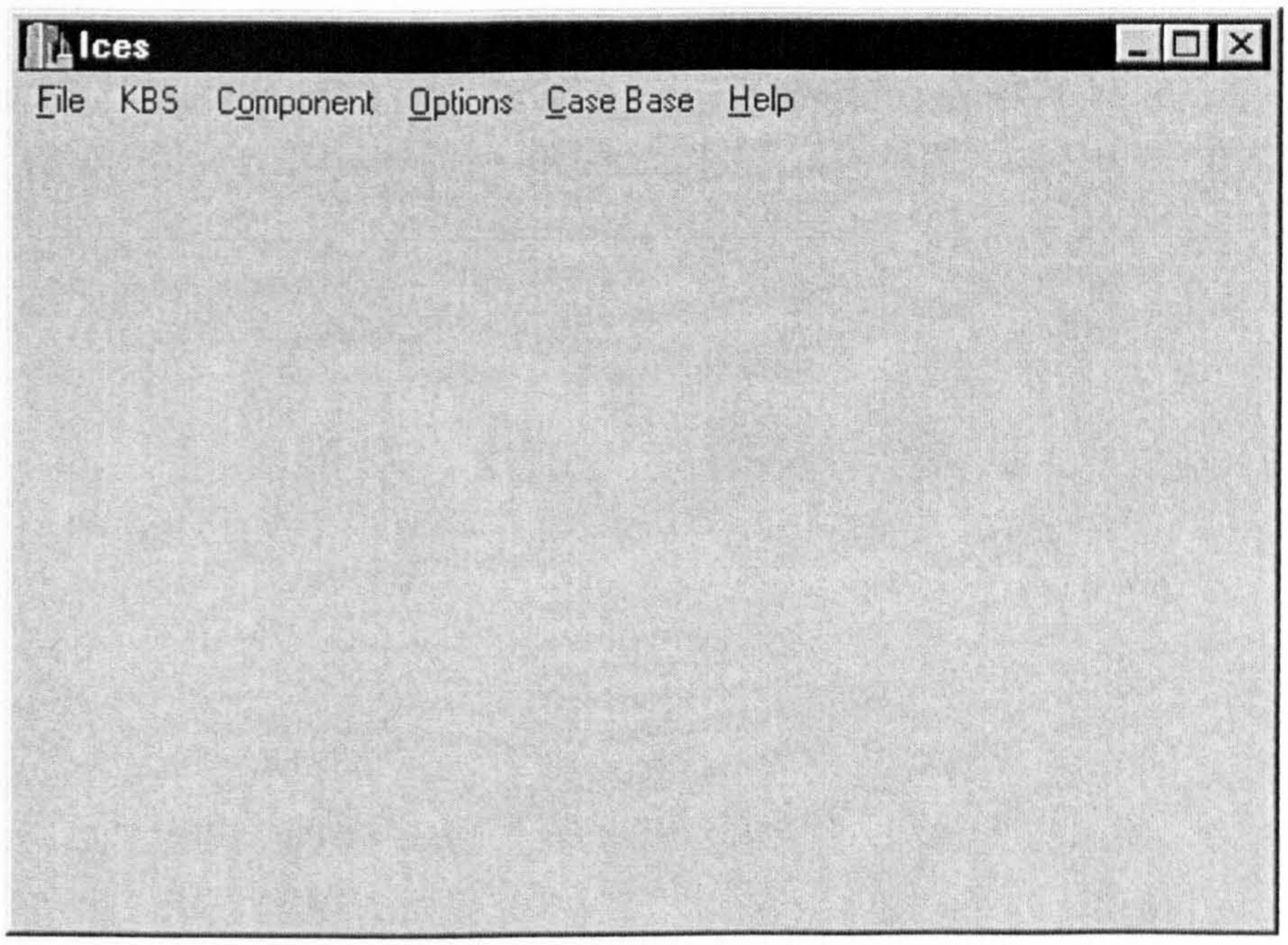



Figure 9.2, ICES Main Menu

9.3. This dialog box has one edit box residing in it. The user should enter the component which it is desired to design in this edit box. The entry of the component will enable some or all of the menu options residing beneath the KBS main menu option. In relation to the methodology described in Chapter 7, this relates to the manufacturing and structures rule bases as outlined in section 7.2.4, where the system requires that the user identify the component of interest. The code that supports the Enter Component menu option is explained in section C.2.2 of appendix C. In addition, this appendix explains how the code facilitating the operation of the Structure Identification dialog could be expanded to support the entry of other aircraft components apart from a foreplane.

#### 9.2.1.3. KBS

At start-up, all the pull-down menu options in the KBS menu are disabled by default. As indicated above, the entering of an appropriate component i.e., a descriptive string, in the Structure Identification dialog box will enable the pull-down menu options in the KBS i.e., the user will see the menu options become non-grey and active and the operations that the menu options support may now be accessed. Section C.2.3 of appendix C explains the operation of the code that facilitates the enabling of the KBS menu options.

##### 9.2.1.3.1. KBS Menu Options - Design Driver Selection

Design Driver Selection is the first KBS menu option. It provides access to a further sub-menu, see Figure 9.4. This sub-menu is disabled on application start-up but is enabled in conjunction with the other KBS menu options. The Design Driver Selection sub-menu presents the user with a series of generic terms common to aircraft operation, see Figure 9.4. In addition, there are menu options that relate to the broader company view of design i.e., Core-Competences and Cost. It is important to appreciate that as these menu options relate directly to the meta rule base, as described

in section 7.2.1 of Chapter 7, only those sub-menu options that represent the

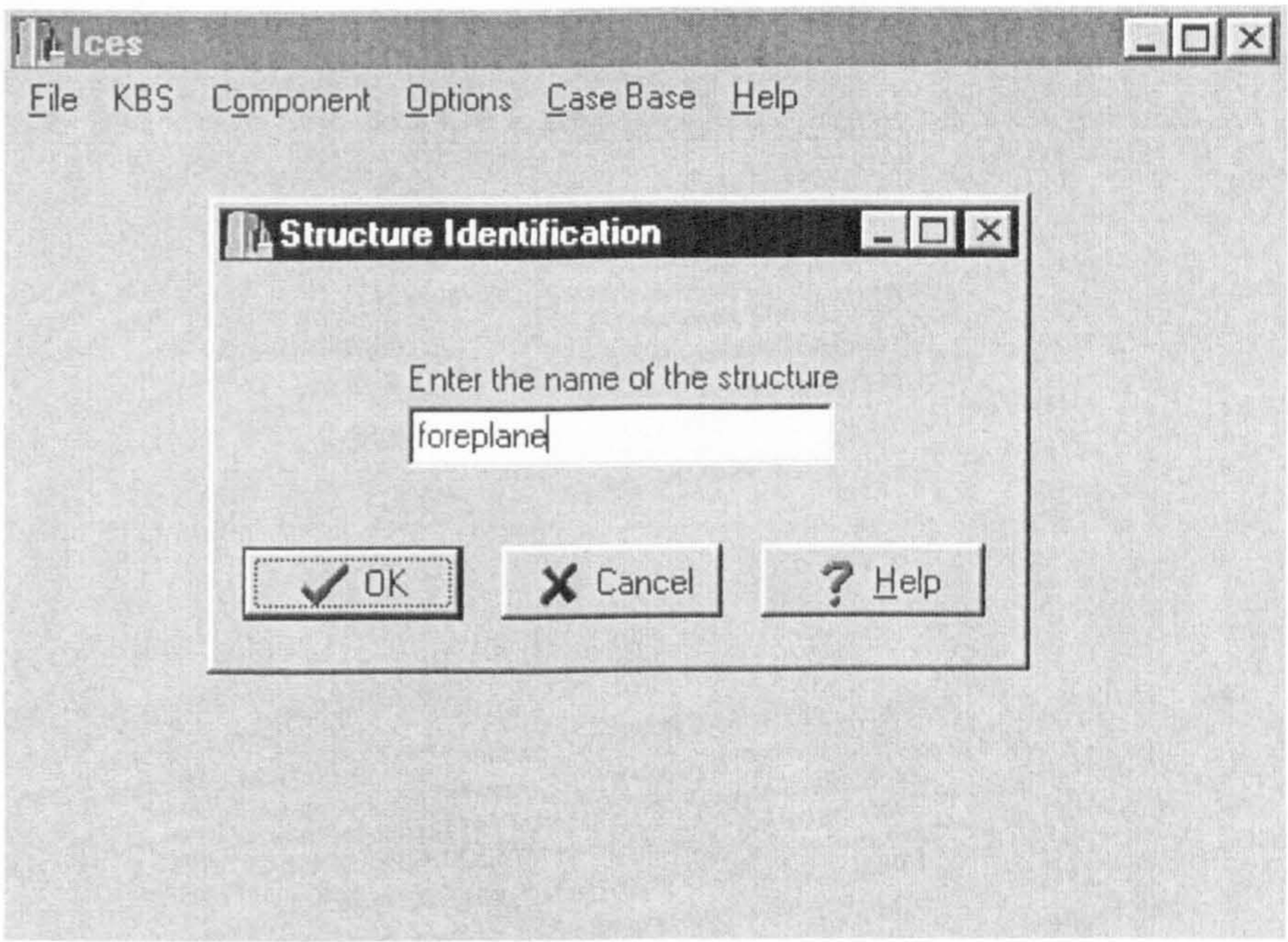



Figure 9.3, Structure Identification Dialog Box

principle operational characteristic features required by the design requirement will be available. For example, if the ICES prototype where to be expanded to accommodate the wing structure as outlined in section C.5.1 of appendix C, the Agility sub-menu option will remain disabled i.e., greyed out. This is because a wing does not by its nature provide agility for an aircraft.

Clearly, if it is required to further expand the ICES prototype such that it can accommodate other aircraft structures it will be necessary to be able to present the user with additional generic terms. These again will be common to aircraft operation but relate to these other aircraft structures. A first step to enable the ICES prototype to handle other aircraft structures is the capability to increase the menu options in the Design Driver Selection Sub-menu. Section C.5.2 of appendix C outlines how menu options may be added to the application.

The selection of the Design Driver Selection Sub-menu options i.e., Agility, Supersonic, Subsonic, Core-Competences and Cost, activate a series of dialog boxes. The code that supports the activation of these dialog boxes is presented in section C.2.3.1 of appendix C. These dialog boxes match the generic terms of the menu options to a set of meta rules and thereby provide access to the design drivers which act on the design process. The full operation of these dialog boxes is discussed in section 9.2.2.2.

#### 9.2.1.3.2. KBS Menu Options - Questions

Questions is the second KBS menu option. The purpose of this menu option is to obtain further details with respect to the component it is proposed to design i.e., as entered in the Structure Identification dialog box. When the user clicks on this menu

option the Questions dialog box is presented. This dialog box asks the user a series of questions that relate to the component in question. As can be seen from Figure 9.5, the

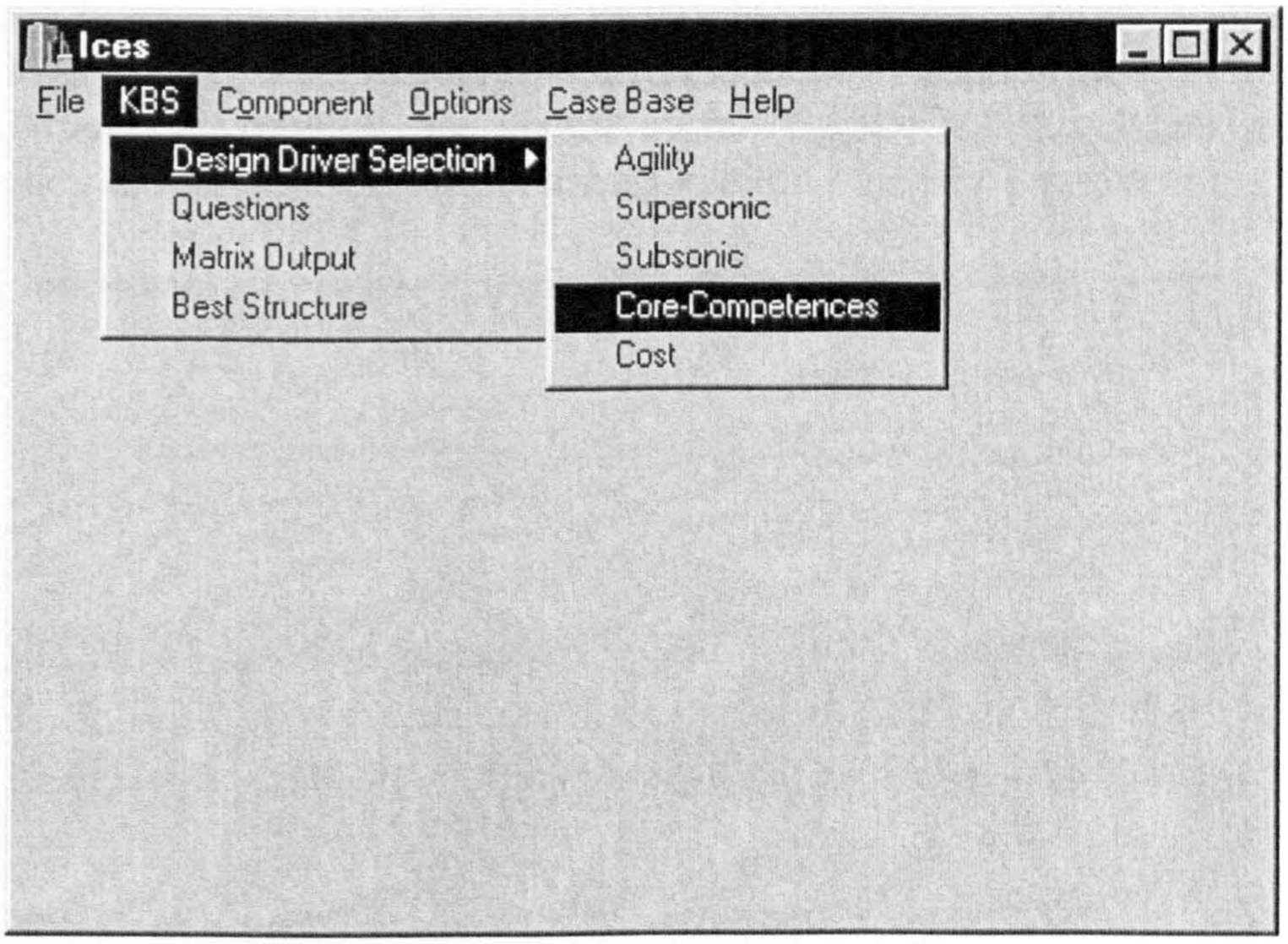



Figure 9.4, The KBS Menu Option

Questions dialog box asks seven direct questions. Edit boxes down the right-hand side of the dialog box enable the user to input his or her responses. It should be noted that apart from the usual dialog box buttons i.e., OK, Cancel and Help, there is also a Reset push button. As its name implies this push button permits the user to cancel previous answers to the questions and start again if so desired.

As already indicated, when the user clicks on the Questions menu option the Questions dialog box is presented. The user should then answer the questions presented in the manner requested by each particular question. This will be either *y* or *n* for yes or no, or a numerical value. Once all the questions have been answered the user should click on the OK push button. The code that supports the operation of the Questions dialog box is provided in section C.2.3.2 of appendix C.

Clearly, to expand the ICES prototype as it has been developed within this study it would be desirable to be able to ask the user additional questions. With this in mind section C.5.3 of appendix C provides the reader with an explanation of how to expand the existing code such that it facilitates the asking of such questions.

#### 9.2.1.3.3. KBS Menu Options - Matrix Output

On selecting the Matrix Output menu option from the KBS menu the system presents the user with the Design Driver Ranking Matrix Output dialog box, see Figure 9.6. The principle purpose of this dialog box is to derive all the importance ratings for the design drivers acting on the design process. The reader is directed to section 7.2 of Chapter 7 which discusses in detail the KBS underlying methodology and in particular in the context this discussion, the derivation of the design driver importance ratings.

In order to derive the design driver importance ratings the Design Driver Ranking Matrix Output dialog box is the focal point for two sources of information. Firstly, the mean average impaction scores for each of the design drivers. Secondly, the design driver sensitivity scores. The method by which the mean average impaction scores and the design driver sensitivities are obtain in the context of this ICES prototype implementation are discussed in sections 9.2.1.4.1 and 9.2.1.4.2 respectively.

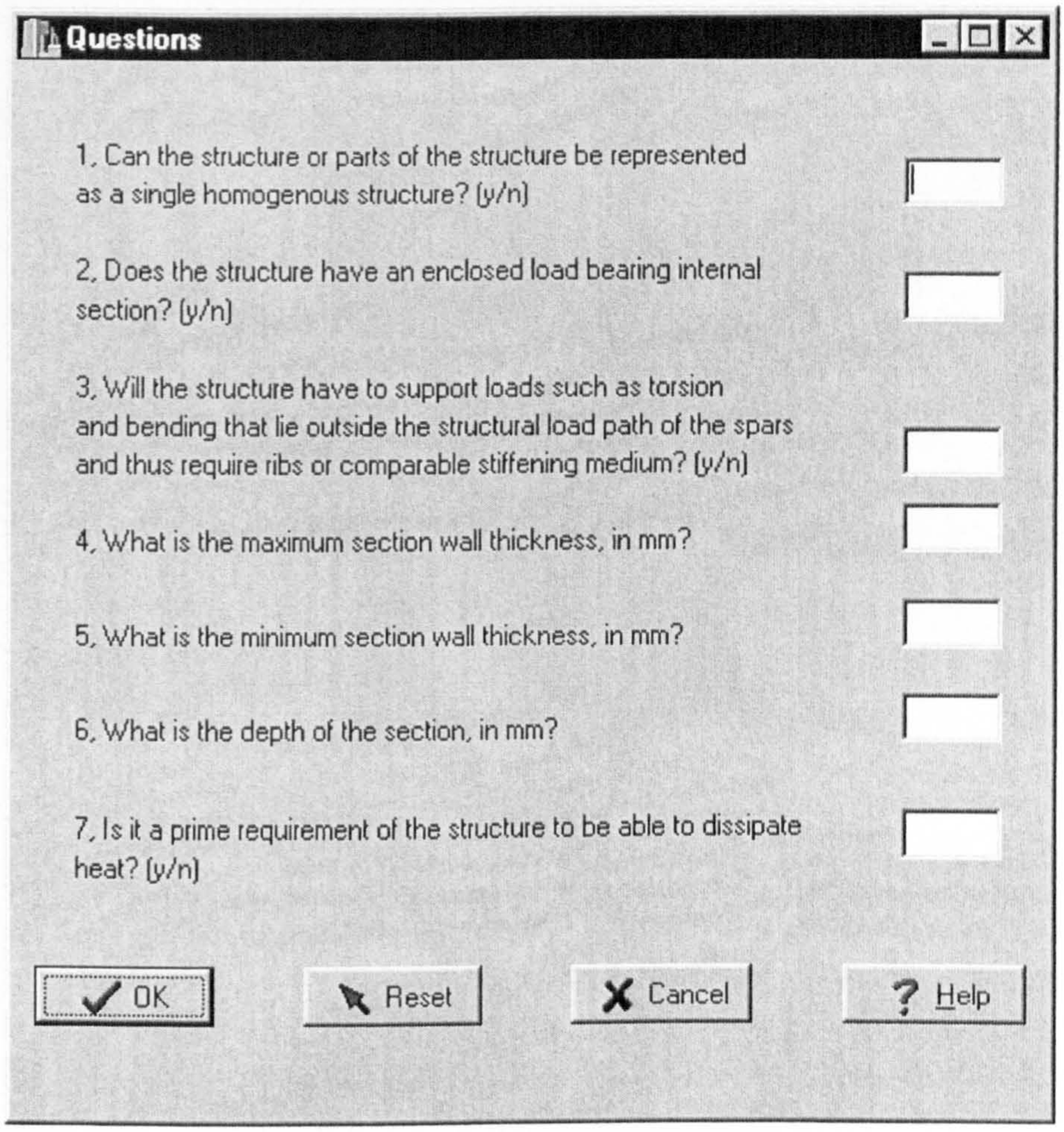



Figure 9.5, Questions Dialog Box

As can be seen from Figure 9.6, against each of the design drivers listed in the Design Driver Ranking Matrix Output dialog box there is a row of five edit boxes. Above these edit boxes are a series of headings which indicate the output which each of the edit box displays i.e., Average, Sum, Updates, Sensitivity and Importance Rating. Taking these headings in order; the Average edit box displays the current mean average of the impaction scores for the adjacent design driver; the Sum edit box displays the sum of the mean averages entered in the Average edit box divided by the number of updates; the Update edit box indicates the number of times any particular design driver mean impaction score has been updated; the Sensitivity edit box indicates the sensitivity of the design driver i.e., how quickly a change in the design driver will impact on the design. By taking the value in the Sum edit box and dividing it by the value in the Update edit box; multiplying this by the value in the Sensitivity edit box and taking the overall square route gives the value in the

Importance Rating edit box. The code that supports the operation of the Design Driver Ranking Matrix Output dialog box is explained in section C.2.3.3 of appendix C.

Clearly, to expand the ICES prototype as it has been developed within this study it would be desirable to be able to add additional design drivers and associated edit boxes to the Design Driver Ranking Matrix Output dialog box. With this in mind section C.5.4 of appendix C provides the reader with an explanation of how to expand the existing code such that it is possible include additional design drivers.

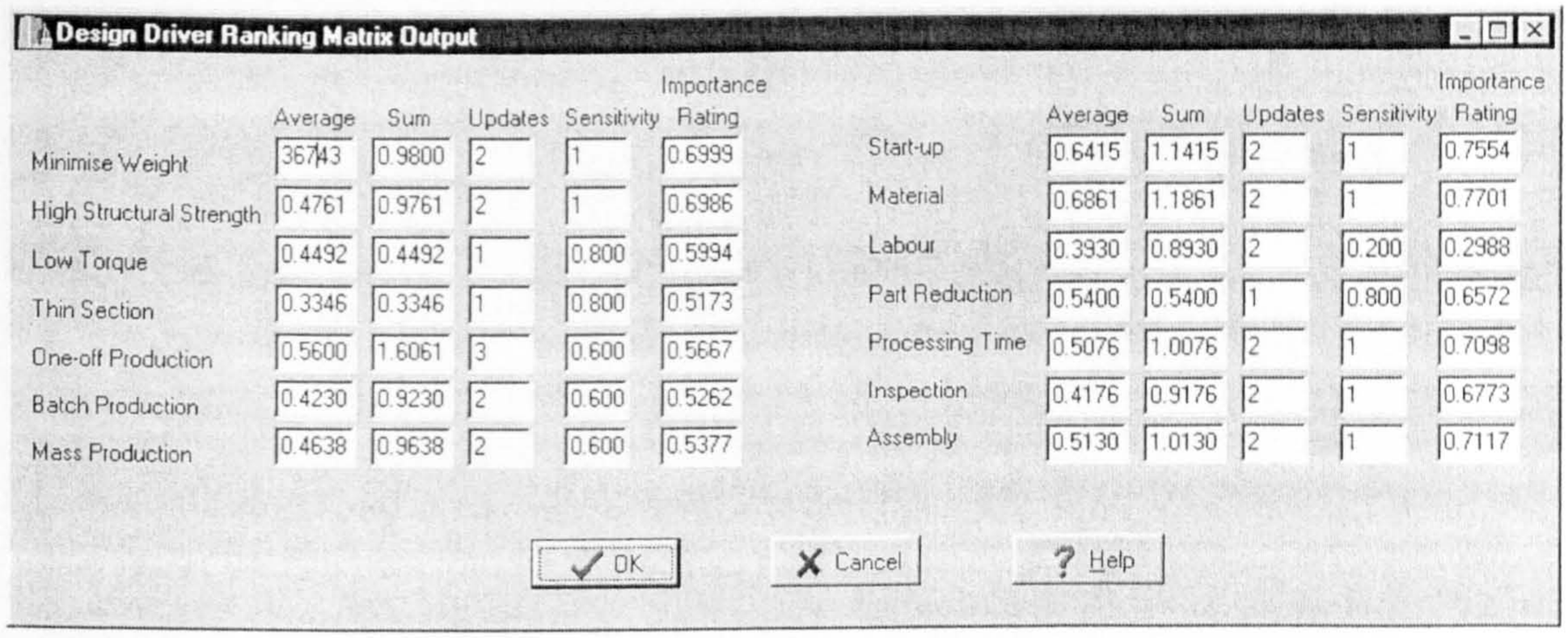



Figure 9.6, Design Driver Ranking Matrix Output Dialog Box

#### 9.2.1.3.4. KBS Menu Options - Best Structure

When the Best Structure menu option is selected from the KBS menu, the system presents the user with the Best Structure dialog box. This dialog box provides the necessary user interface controls to permit the user to determine what the KBS component of the ICES prototype considers to be the 'best structure'. In the context of describing the operation of the ICES prototype, a detailed explanation of the operation of the Best Structure dialog box is given in section 9.2.2.4 when the KBS component of the ICES prototype is discussed. The purpose of this section is to provide the user with an appreciation of where this important feature of the ICES KBS resides in the context of the overall ICES prototype 'front-end'.

### 9.2.1.4. Options

Selection of the Options menu option reveals two pull-down menu options. These being the Design Driver Matrix Update and Design Driver Sensitivities Update menu options. Both of these menu options provide the facility to update the information residing in the Design Driver Ranking Matrix Output dialog box as discussed in section 9.2.1.3.3; specifically the mean values of the impaction scores acting on the design drivers and the individual sensitivities of each of the design drivers. These menu options are now discussed in greater detail below.

#### 9.2.1.4.1. Options Menu Options - Design Driver Matrix Update

The selection of the Design Driver Matrix Update menu option reveals a list of further sub-menus. Each of the these sub-menus relates to one of the design drivers embraced by the ICES prototype, see Figure 9.7. Selecting anyone of these menu options will present a further sub-menu which has two menu options, Default and Default Update. These two menu options present the user with two almost identical dialog boxes. The

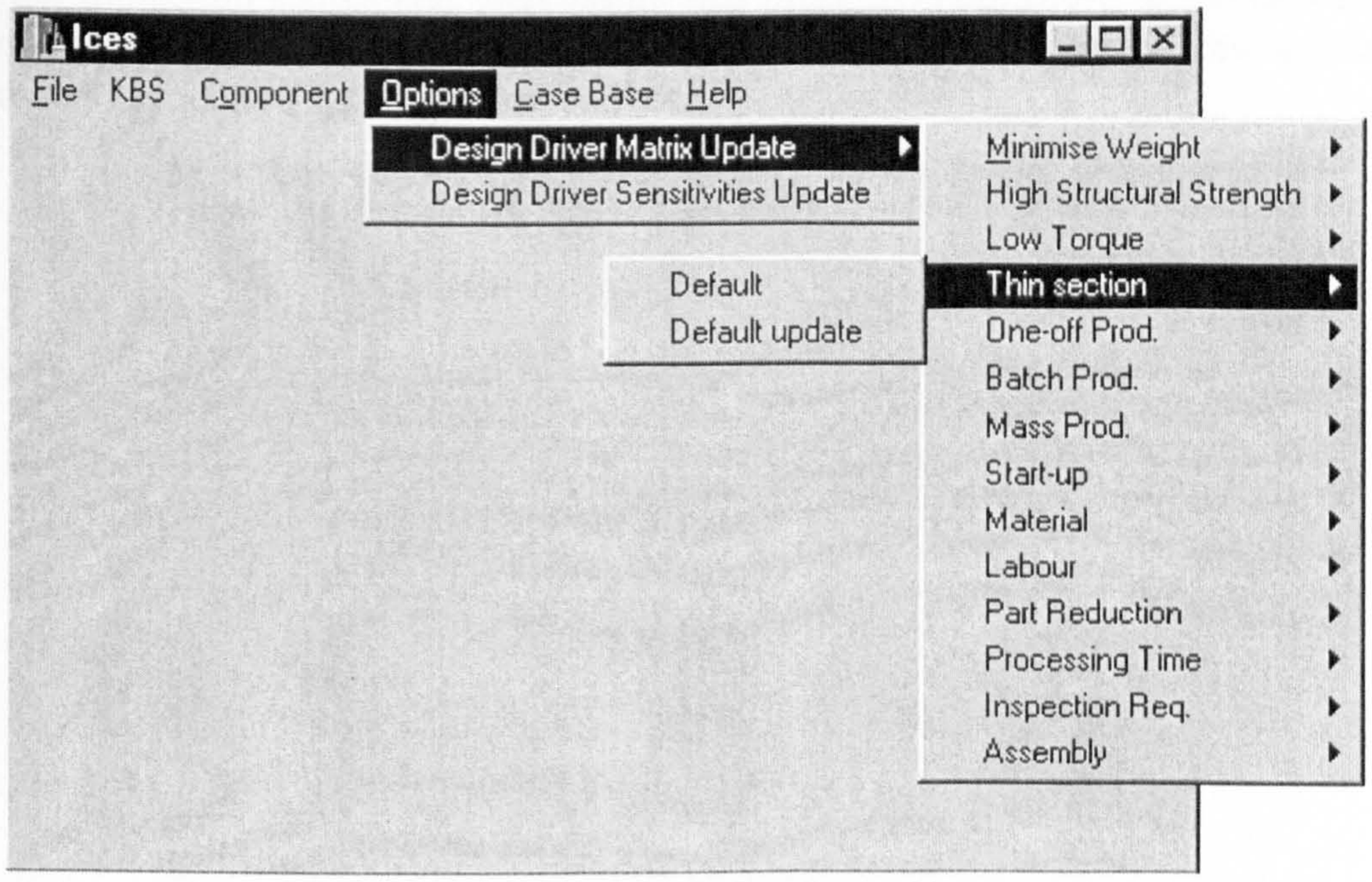



Figure 9.7, The Design Driver Matrix Update Menu Option

dialog boxes show in numerical terms the impact that one particular design driver has on all the other design drivers operating on the system. The reader is referred to section 7.2.2.1 of Chapter 7, which discusses the methodology supporting the design driver matrix, in particular reader should examine Figure 7.2 which shows a pictorial representation of the design driver matrix. The information presented in each of the dialog boxes discussed here represents one row in this matrix.

The Default menu option presents the user with the Default dialog box, there being one for every design driver. This dialog box presents the user with the current impaction scores for a particular design driver. The numerical values residing in the edit boxes of this dialog box cannot be edited. There purpose is for reference only.

The Default Update menu option presents the user with the Default Update dialog box, again there being one for every design driver as with the Default dialog box. The Default Update dialog box for the minimum weight design driver is illustrated in Figure 9.8. The Default Update dialog boxes for all the other design drivers acting on the system are of an identical format. The Default Update dialog box differs from the Default dialog box in that it permits the user to enter new values in the edit boxes. These values correspond to the degree by which the design driver, after which the dialog box is named, impacts on the other design drivers; scores in the range of 0 to 1 are entered in the appropriate edit boxes. Again, the reader is referred to section 7.2.2.1 of Chapter 7 where the design driver ranking technique (DDRT) is discussed

in detail. This provides guidance with respect to the how the scoring of the impact of one design driver on another may be approached. As can be seen from Figure 9.8, the Minimum Weight Default Update dialog box has four control push buttons. It is considered that the operation of the OK, Cancel and Help push buttons is self explanatory. However, the Update push button requires explanation. The purpose of the Update push button is to initiate the calculation of the mean average of the design driver impaction scores and then assign this mean average score to the to the Design Driver Ranking Matrix Output dialog box. The code that supports this operation is provided in section C.2.4.1 of appendix C.

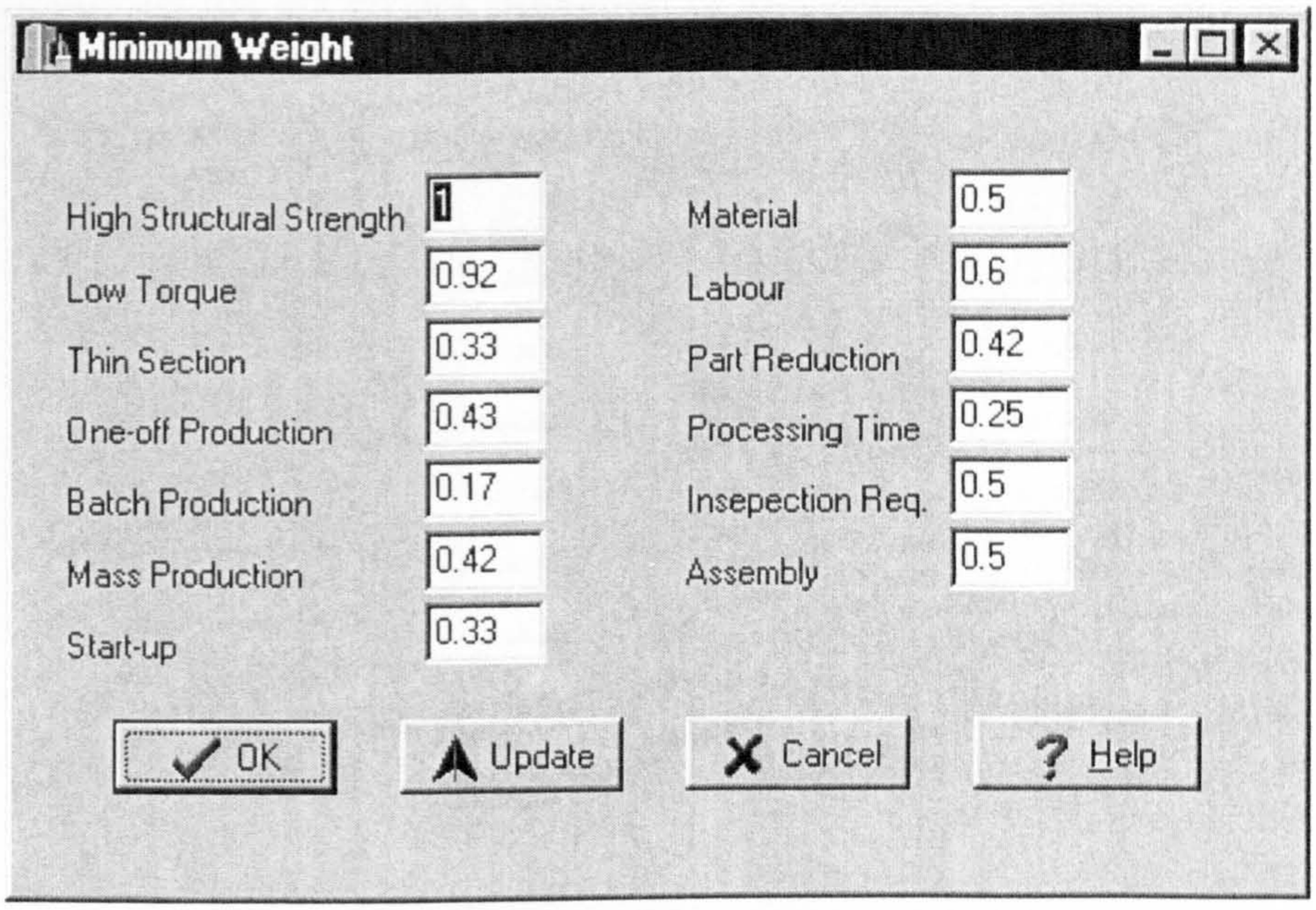



Figure 9.8, Minimum Weight Default Update Dialog Box

The addition of further design drivers is a significant area where the ICES prototype could be expanded. In this context, section C.5.5 of appendix C provides details with respect to copying a Default Update dialog box to the C++ Builder Object Repository such that it can be called up, edited and subsequently used again for a new design driver. In addition, section C.5.5 provides the reader with an explanation of how to expand the existing supporting code such that a new Default Update dialog box can be appended to the current ICES prototype.

#### 9.2.1.4.2. Options Menu Options - Design Driver Sensitivities Update

When the Design Driver Sensitivities Update menu option is selected from the Options menu the Sensitivity dialog box is presented to the user, see Figure 9.9. The purpose of this dialog box is to enable the user to indicate the sensitivity of each of the design drivers acting on the system. Specifically, numerical weightings are applied to design drivers depending on how much they can be changed with respect to how strongly a change in the design driver acts on the design. The reader is referred to section 7.2.2.2 of Chapter 7 where the methodology underlying the Sensitivity dialog box is discussed.

The Sensitivity dialog box has fourteen edit boxes, one corresponding to each design driver acting on the system. The user may enter in these edit boxes numerical values ranging from 0.2 to 1; where 0.2 indicates the design driver is completely insensitive to change and 1 indicates the design driver is 'totally' sensitive to change. As can be seen from Figure 9.9, the Sensitivity dialog box has apart from the usual dialog box control push buttons i.e., OK, Cancel and Help, an Update push button. The purpose of this Update control is to assign the numerical values residing in the edit boxes to the corresponding edit boxes listed under the Sensitivity heading in Design Driver Ranking Matrix Output dialog box as discussed in section 9.2.1.3.3. The code supporting the underlying operation of the Sensitivity dialog box is given in section C.2.4.2 of appendix C.

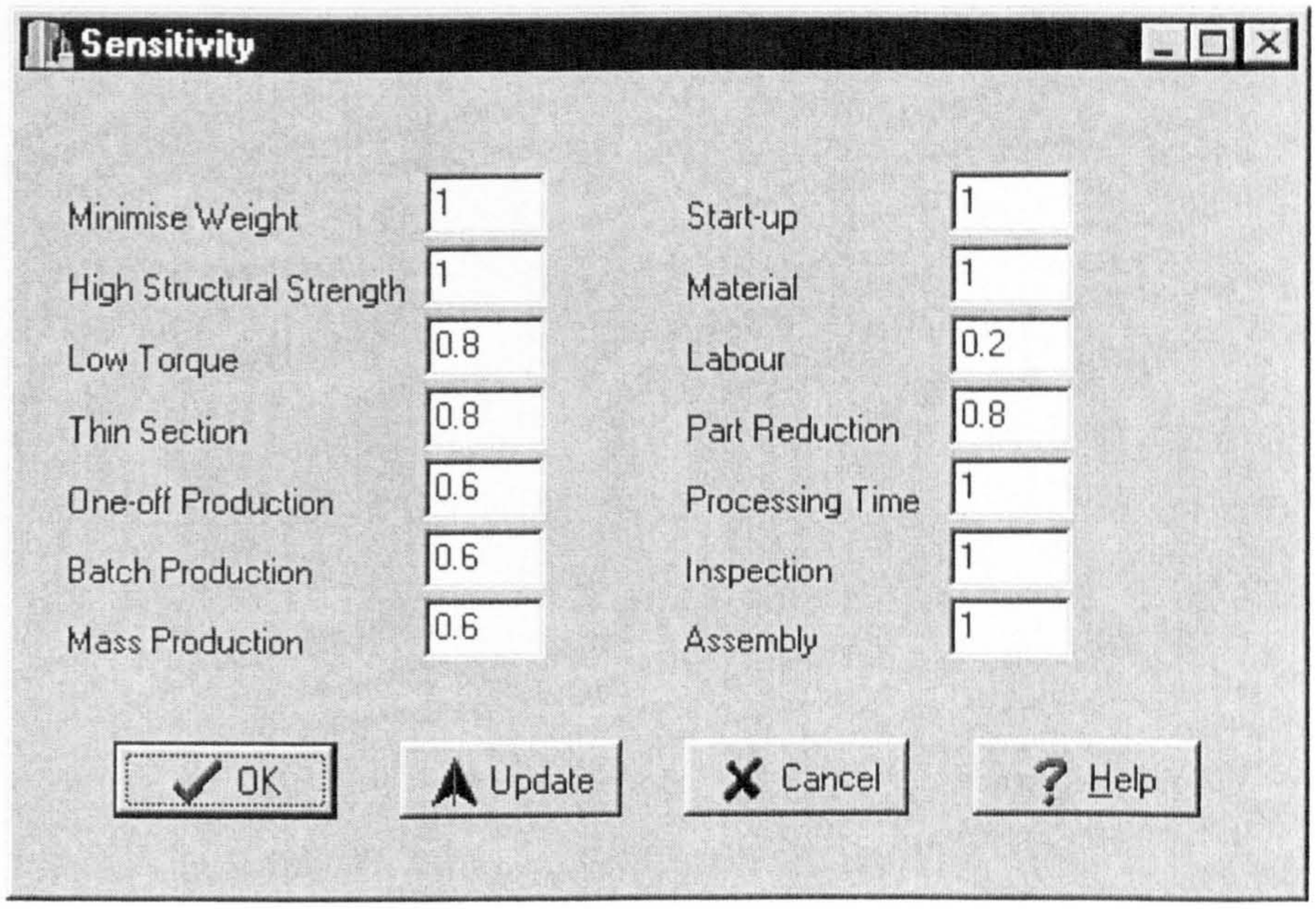



Figure 9.9, Sensitivity Dialog Box

### 9.2.1.5. Case Base

The Case Base main menu option has two menu items, these being Enter Case and Case Query and Jury, see Figure 9.10. In the context of describing the operation of the ICES prototype it is considered appropriate to only give a brief overview of the case base in the following two sections and confine the detailed discussion to section 9.2.3.

#### 9.2.1.5.1. Case Base Menu Options - Enter Case

When the Enter Case menu option is selected the New Case window is presented to the user. It important to note that the C++ Builder development environment does not distinguish between dialog boxes and windows, they are one of the same. However, in the context of this prototype, windows are defined as those dialog boxes which have their own menuing capability. The purpose of the New Case window is to enable the user to enter new cases into the case base. The New Case window also provides full case editing capability. The detailed operation of the New Case facility is discussed in section 9.2.3 where the full operation of the ICES case base is discussed.

#### 9.2.1.5.2. Case Base Menu Options - Case Query and Jury

On selecting the Case Query and Jury menu option from the Case Base menu the system presents the user with the Case Query and Jury window. The principle purpose of this window is to enable the user to query the cases residing in the case base. In addition, the Case Query and Jury window facilitates the operation of the jury system as outlined in section 7.3.3 of Chapter 7. Further, the Case Query and Jury window provides the capability to apply adaptation to the cases residing in the case base. As with the previous section, the full operation of the Case Query and Jury window of the ICES case base is described in detail in section 9.2.3.

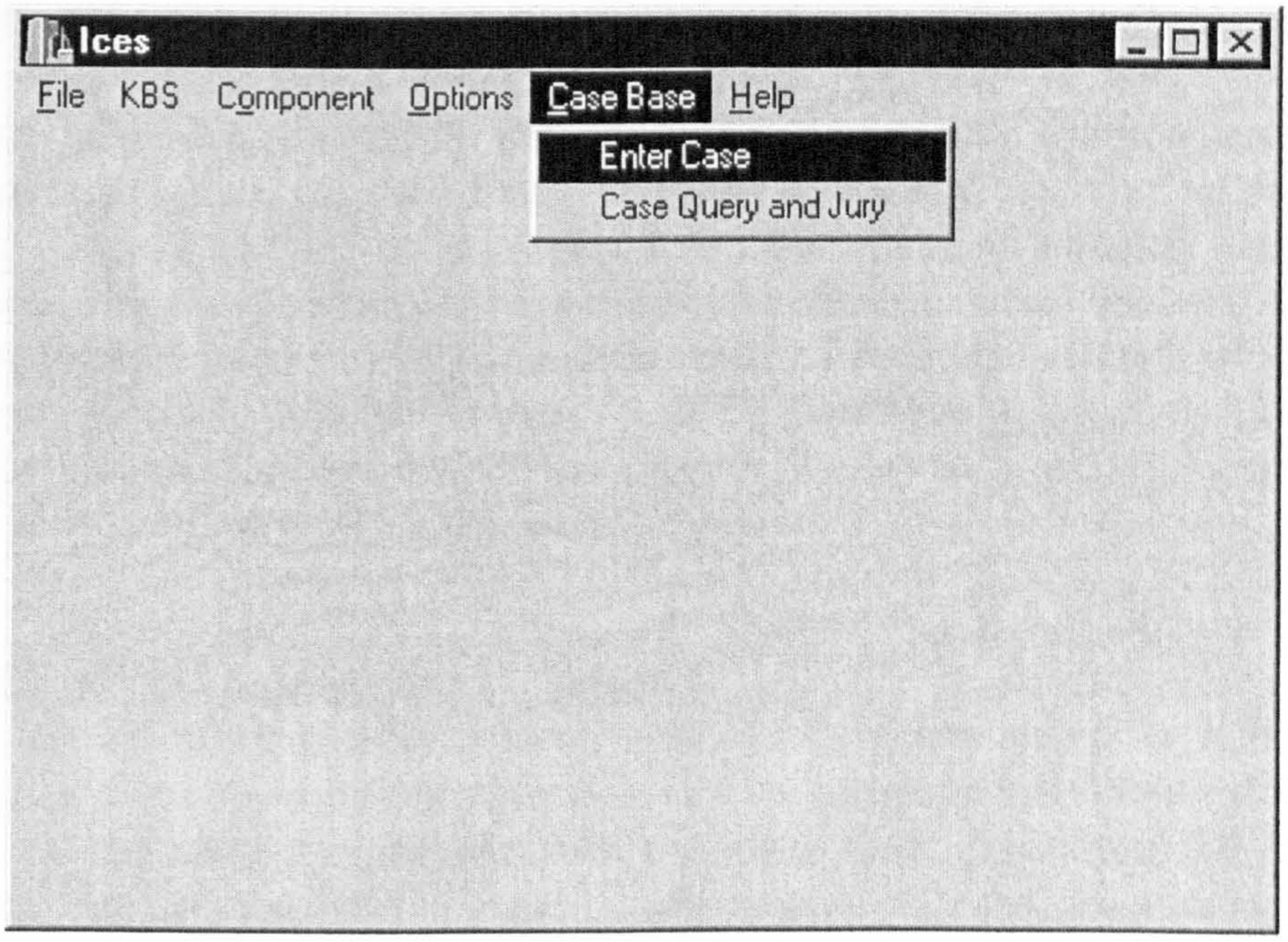



Figure 9.10, The Case Base Menu Options

### 9.2.1.6. Help

The Help main menu option has three menu items, these being KBS Operation, Case Base Operation and About.

#### 9.2.1.6.1. Help Menu Options - KBS Operation

Selecting the KBS Operation option from the Help menu causes the system to present the user with a KBS help dialog box. This dialog box provides the user with guidance with respect to how to run the KBS capability of the ICES prototype.

#### 9.2.1.6.2. Help Menu Options - Case Base Operation

In a similar manner to the KBS Operation menu option, the Case Base Operation menu option in the Help menu presents the user with a Case Base help dialog box. This provides the user with guidance with respect to how to run the case base capability of the ICES prototype.

#### 9.2.1.6.3. Help Menu Options - About

On selecting the About menu option from the Help menu, the system presents the user with the About dialog box. This dialog box presents details concerning the ICES prototype developer, the version number, date of construction and where the prototype was built.

### 9.2.2. KBS Operation

This section will now outline in a logical manner, the operation of the ICES prototype KBS capability. Where appropriate reference will be made to the ICES software 'front-end' components introduced previously in this chapter. In addition, reference will be made to the ICES methodology as documented in Chapter 7. The explanation of the KBS capability will be supported with code examples provided in the appropriate sections of appendix C. The areas of the KBS which lend themselves to further development are commented on and the reader is directed again to the appropriate section of appendix C where directions for expanding the code are given. As a final point, it necessary to re-emphasis that the foreplane aircraft structure was the structure selected to drive through the ICES prototype in order to validate the underlying methodology as defined in Chapter 7. As such, all the explanations provided in this section relate to this aircraft structure.

#### 9.2.2.1. Identify the Component of Interest

The first step necessary in order to identify the 'best structure' using the ICES KBS is to declare the aircraft component of interest i.e., the component it is desired to design. The user should select Component from the main menu, and from this the Enter Component menu option. This now permits the user to enter the structure name in the Structure Identification dialog box as discussed in section 9.2.1.2.1. The name of this structure in this instance being foreplane. The action of entering the foreplane structure in the Structure Identification dialog box has the effect of enabling the KBS menu options, this aspect was discussed in sections 9.2.1.3 and 9.2.1.3.1.

#### 9.2.2.2. Meta Rule Base

Having enabled the KBS menu options through the entry of the foreplane structure in the Structure Identification dialog box, the user should access the Design Driver Selection menu option from the KBS menu. This menu presents a further menu list which presents to the user those generic terms common to aircraft operation that relate to what the foreplane structure provides for the aircraft i.e., agility, supersonic performance, subsonic performance. In addition, to the Agility, Supersonic and Subsonic menu options, the menu list also includes menu items which relate to the broader company view of design i.e., Core-Competences and Cost, see Figure 9.4. The system having enabled those generic menu options which relate to what the foreplane structure provides for an aircraft, requires that the user now select these menu options in turn. In so doing, the user will be presented with a series of dialog boxes that match the generic terms declared by the menu items to a set of generic meta rules. These identify the fundamental relationship between the generic terms common to aircraft, those relating to the broader company view, and what is physically required to achieve the generic terms. The process of defining what is physically required to achieve these

generic terms facilitates the defining of the design drivers that act on the design process.

Taking the menu options in order. The selection of the Agility menu option presents to the user the Agility dialog box as illustrated in Figure 9.11. As can be seen from this figure there are four check boxes corresponding to four design drivers, adjacent to which there are four note pad push buttons. The note pad push buttons when selected inform the user via a dialog message box of the meta rule that will be fired if the user clicks on the corresponding check box. Figure 9.12 shows the dialog message box for the minimum weight design driver. Whilst, the generic menu driven agility term declares the potential for firing four meta rules and thus, introducing four

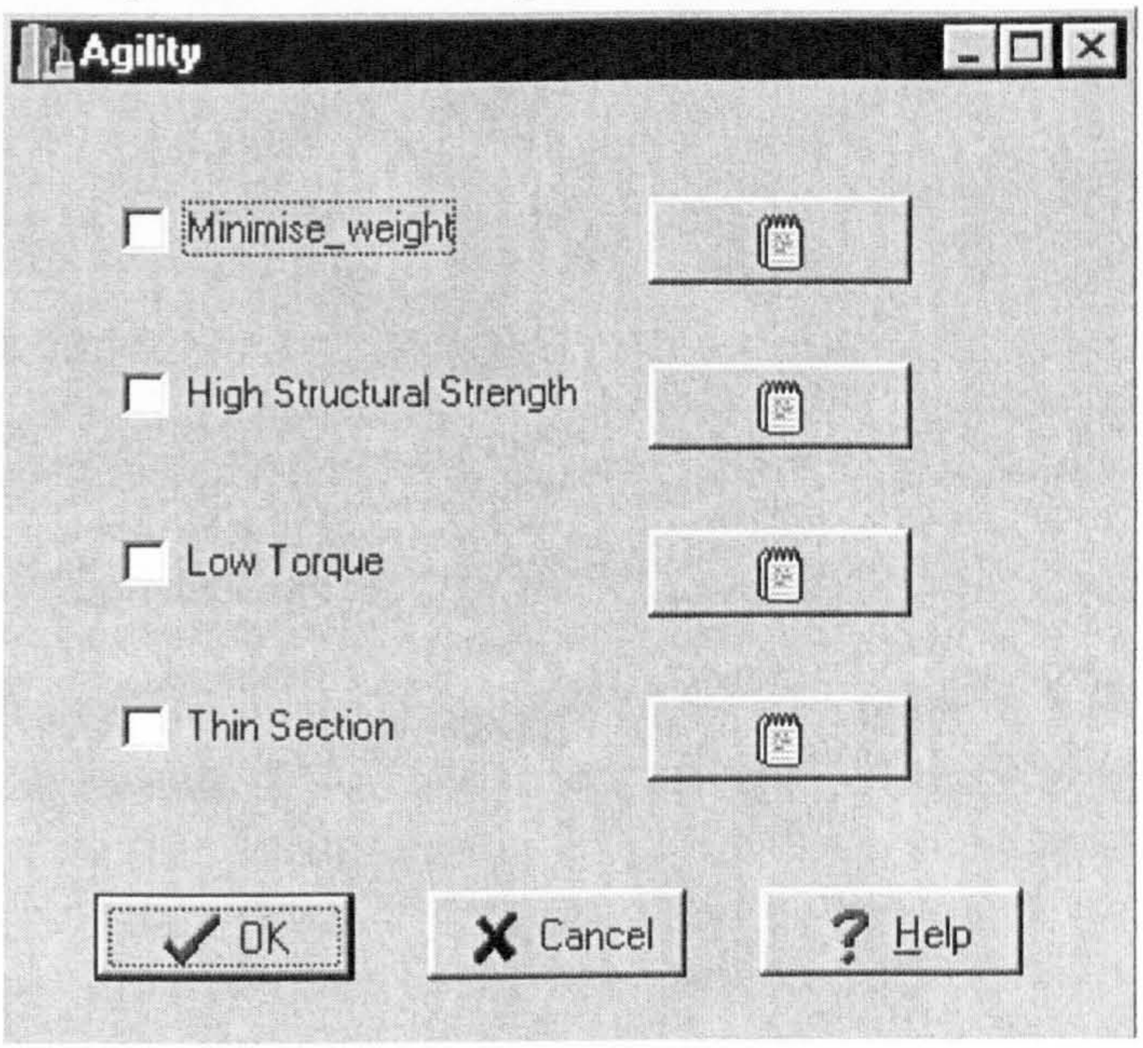



Figure 9.11, The Agility Dialog Box

design drivers into the design process, it is the user who decides whether the meta rule is appropriate to the design problem in question. Clearly, those design drivers introduced into the design process by clicking on the check boxes in the Agility dialog box must be made accessible to those down-stream components of the KBS. The code that supports this capability is explained in section C.3.1 of appendix C.

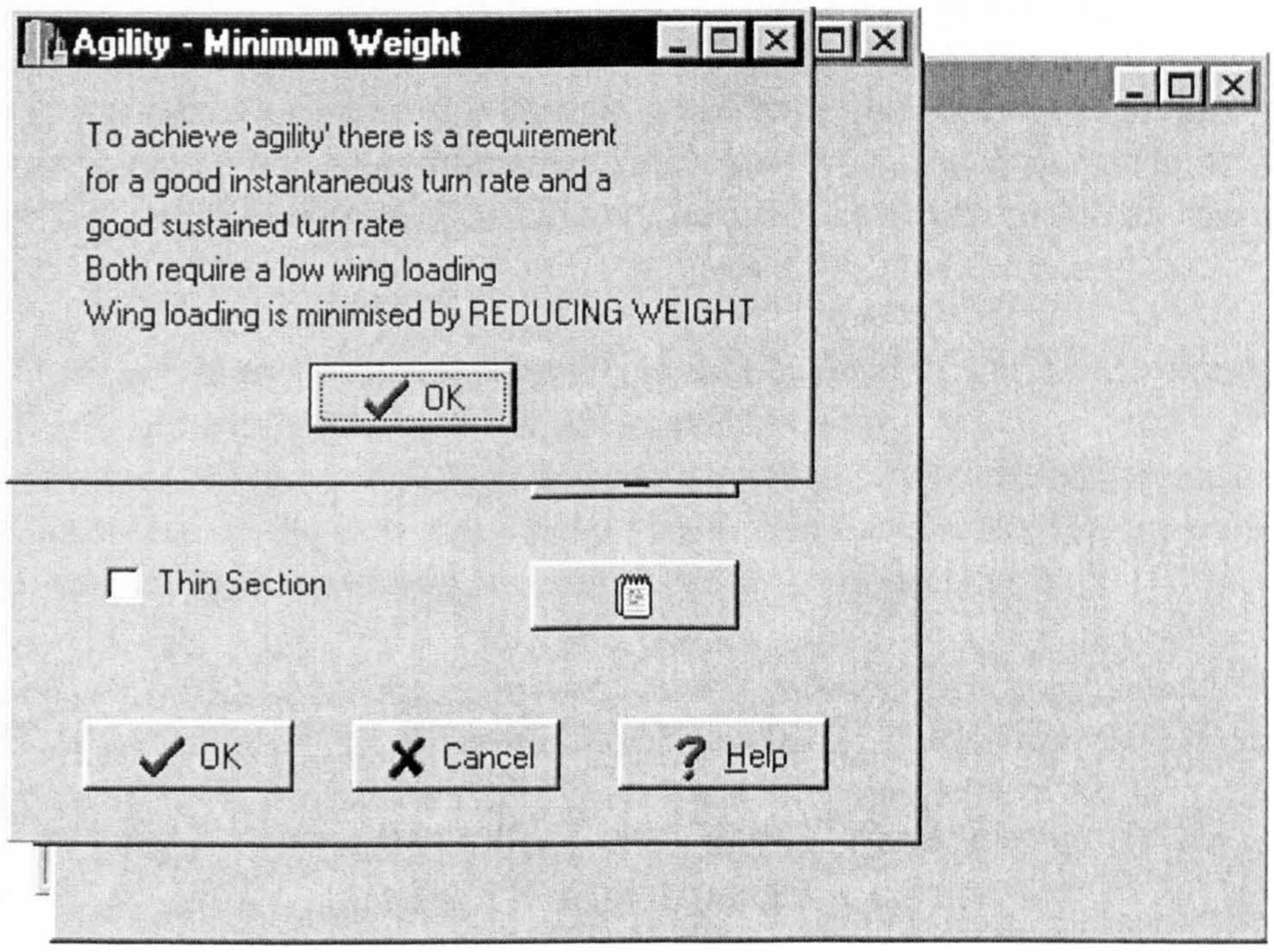



Figure 9.12, The Dialog Message Box for the Minimum Weight Design Driver

The Supersonic and Subsonic menu options when selected, present the user with the Supersonic and Subsonic dialog boxes respectively. These dialog boxes operate in an identical manner to the Agility dialog box. However, the meta rules declared in these dialog boxes naturally relate to the generic terms supersonic and subsonic. Both of these dialog boxes have just one check box each, corresponding to the thin section design driver. However, the dialog message boxes differ in there context. The Supersonic dialog message box states that to alleviate the effects of shock stall and postpone the drag rise to high Mach numbers there is a requirement for thin sections. The Subsonic dialog message box, on the other hand, states that at high speeds thick sections are impractical because the excessive airflow acceleration over the surface produces velocities which exceed the local sonic velocity and lead to premature shock wave formation. Hence, there is a requirement for thin sections. So both the supersonic and subsonic generic menu terms lead to the introduction of the thin section design driver into the design process but from two different directions.

As with the Agility dialog box, the design drivers introduced into the design process by clicking on the check boxes in the Supersonic and Subsonic dialog boxes must be made accessible to the down-stream components of the KBS. This is achieved in an identical manner as described for those design drivers declared within the Agility dialog box. Again the reader is directed to section C.3.1 of appendix C for an explanation of how the underlying code provides this capability.

The Core-Competences menu option is one of those menu options which relates to the broader company view of design. Specifically, the impact of research and development (R&D) on the design process. On selecting the Core-Competences menu option the user is presented with the Research and Development Input dialog box, as illustrated in Figure 9.13. It can be seen from this figure that the user is required to

select one of three possible levels of R&D i.e., by clicking on the appropriate radio button. As commented upon in section 7.2.3 of Chapter 7, the degree of R&D input may be due to a natural design progression or possibly be the result of strategic input by senior management. In order to accommodate this possible wide range of R&D input the user is required to select an appropriate level of R&D based on one of the following:

- The new design is to be developed relying solely on existing core competences (this may or may not include and R&D element).
- The development of the new design will aim to expand existing competences.
- The development of the new design will be looking at developing a totally new structure and thus gain new core competences for the organisation.

The code that relates to the operation of the radio buttons and provides this capability to select between the above mention three levels of R&D is given in section C.3.1 of appendix C. It should be noted that if the user neglects to select a level of R&D input the system will inform the user. This will occur when the user attempts to identify the best structure as discussed in section 9.2.2.4.

Having selected the appropriate level of R&D input, the user should select the Cost menu option. This menu option again relates to the broader company view of design and presents the user with the Cost dialog box, see Figure 9.14. Down the left-hand side of this dialog box there are seven check boxes corresponding to seven design drivers. Adjacent to these check boxes there are seven note pad push buttons. As with the Agility, Supersonic, and Subsonic dialog boxes these note pad push buttons when selected inform the user via a dialog message box of the meta rule that will be fired if the user clicks on the corresponding check box. Again, as with the other dialog boxes,

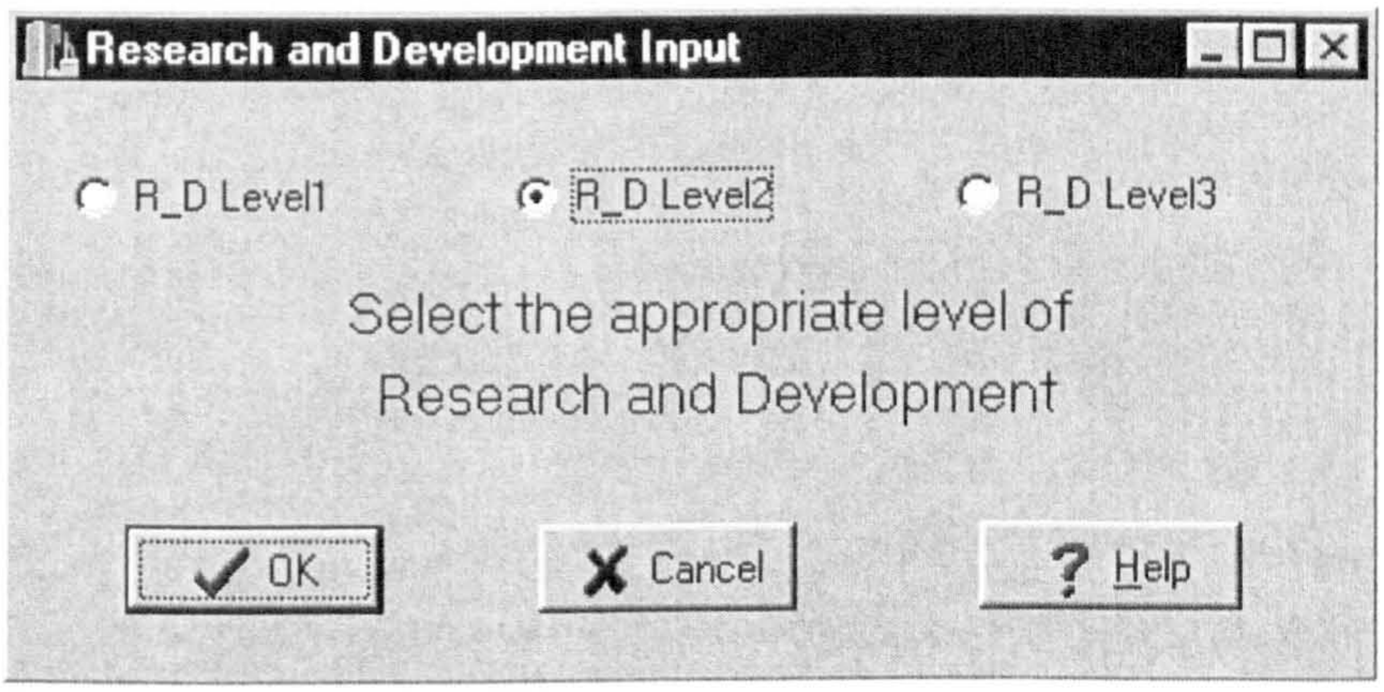



Figure 9.13, Research and Development Input Dialog Box

it is the user who decides whether the meta rule is appropriate to the design problem in question and should click the check box as deemed appropriate. On the right-hand side of the Cost dialog box there are three radio buttons which correspond to the preferred method of manufacture i.e., one-off, batch or mass production. The user should select one of these methods of manufacture. As with the selection of the appropriate level of R&D, if the user neglects to select a method of manufacture the system will inform the user. Again, this will occur when the user attempts to identify

the best structure. The code that supports the underlying operation of the design drivers residing in the Cost dialog box is provided in section C.3.1 of appendix C.

In the context of expanding the ICES prototype, it will be necessary to expand the menu list which presents to the user those generic terms common to aircraft operation as indicated above i.e., as accessed from the Design Driver Selection menu option. In addition to adding further menu options, it will also be necessary to provide additional dialog boxes to facilitate the firing of the associated meta rules. Appendix C.5.6 outlines the process of expanding the ICES prototype with respect to adding further menu options and dialog boxes in these areas.

### 9.2.2.3. Questions

Having fired the appropriate design drivers as indicated in the previous section, the user should now exit the Design Driver Selection menu and select the Questions option from the KBS menu. As indicated section 9.2.1.3.2, when the user clicks on this menu option the Questions dialog box is presented to the user, see Figure 9.5. The operation of the Questions dialog box was discussed in detail in section 9.2.1.3.2. It is sufficient here to say that the user should answer the questions presented in the Questions dialog box and click the OK push button. The reader who at this juncture requires further information is referred back to section 9.2.1.3.2.

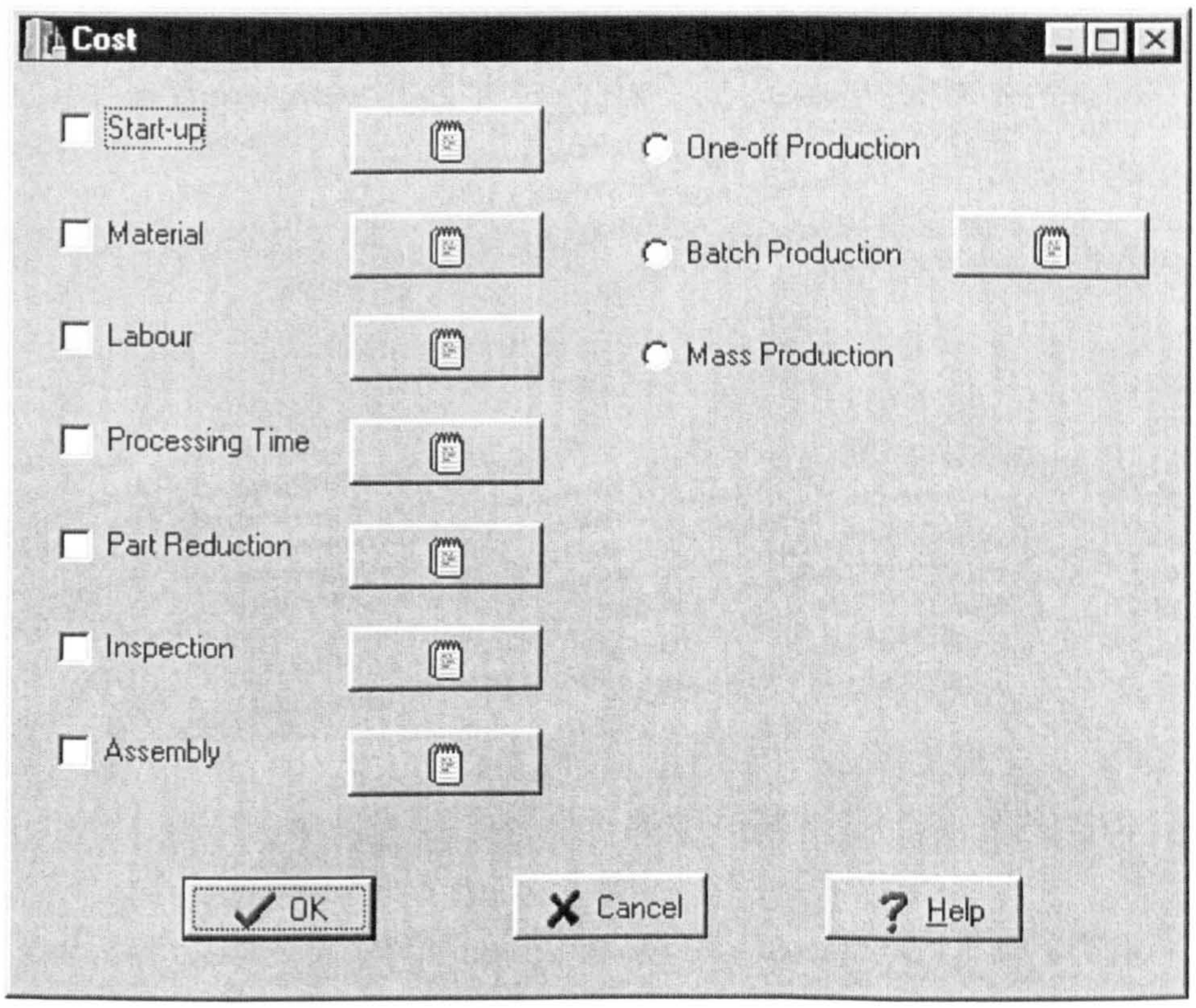



Figure 9.14, The Cost Dialog Box

### 9.2.2.4. The Best Structure

The user having identified the aircraft structure of interest as a foreplane, fired the appropriate meta rules via the Design Driver Selection menu, and answered the questions presented in the Questions dialog box is now in a position to identify the

'best structure'. The user should now select the Best Structure menu option from the KBS menu. The user will now be presented with the Best Structure dialog box, see Figure 9.15. As indicated in section 9.2.1.3.4, which introduced the Best Structure dialog box, the purpose of this dialog box is to tell the user what the KBS component of the ICES prototype considers to be the best structure. Before proceeding further with this explanation the reader is directed to section 7.2 of Chapter 7. This details the operation of the underlying methodology that supports the selection of the best structure. It should be noted at this juncture that the detailed explanation of the operation of the code that supports the underlying functionality of the Best Structure dialog box is provided in section C.3.2 of appendix C.

As can be seen from Figure 9.15, the Best Structure dialog box has two list boxes with scroll bars on the right-hand side. Above these list boxes are the headings 'Manufacturing' and 'Structures', and above these headings is the larger heading 'Ranked Structures'. The list boxes relate to the structures and manufacturing rule bases as outlined in section 7.2.4 of Chapter 7. When the push button 'Best Structure ?' which is situated immediately below the two list boxes is operated, the range of structures that are listed in the two list boxes are ranked in descending order of preference. This is the result of the firing of rules in the manufacturing and structures rule bases. Each rule base's preferred structure is presented at the top of each list box. If the preferred structure in each list box is the same i.e., the two rule bases preferred structures are in agreement, then this structure will be presented in the edit box immediately below the 'Best Structure ?' push button and presented as the best structure. The explanation of the underlying code that facilitates the derivation of the best structure is to be found in sections C.3.2.3 and C.3.2.4 of appendix C.

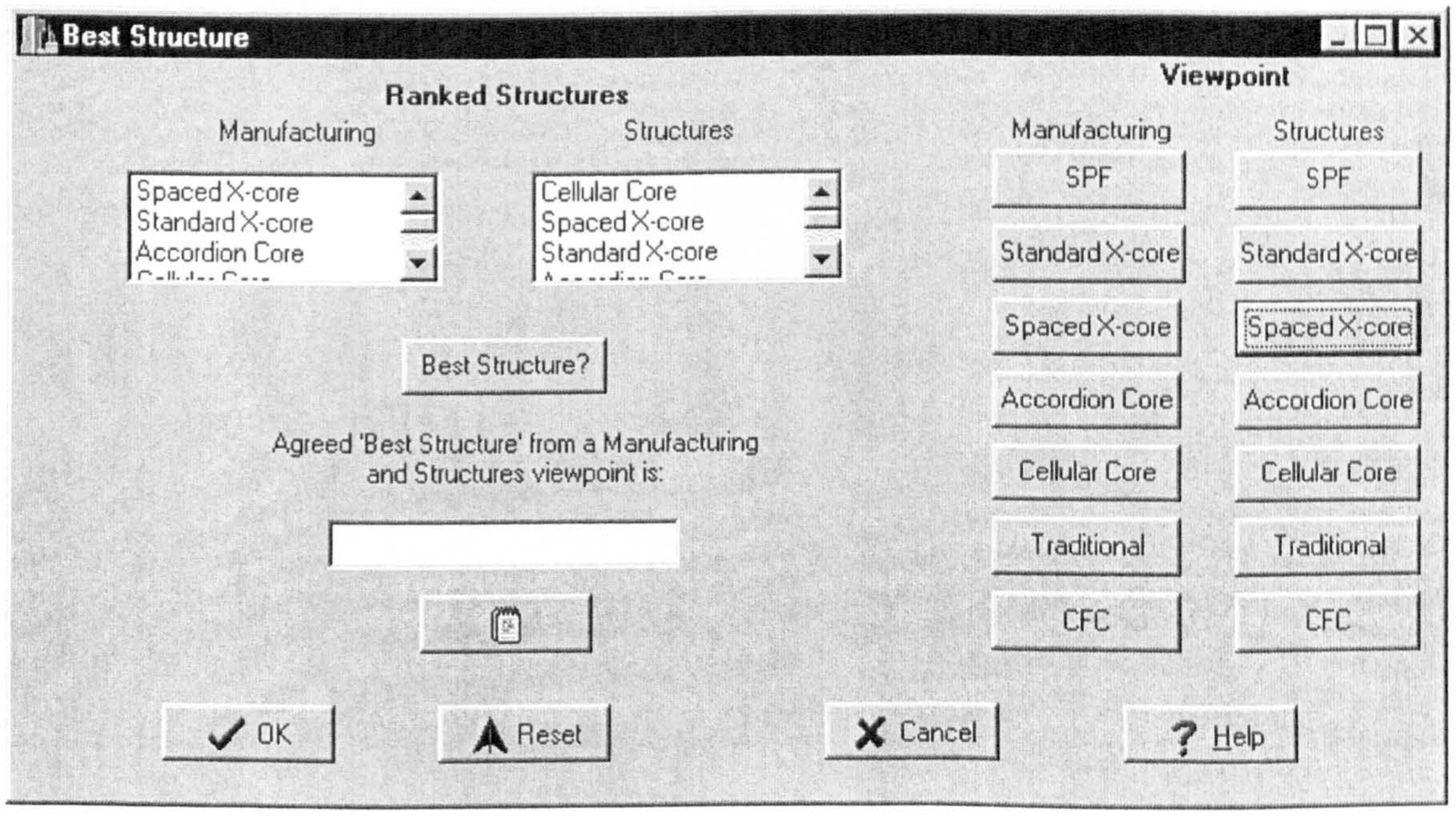



Figure 9.15, The Best Structure Dialog Box

It should be noted that prior to firing of rules in either the structures or manufacturing rule bases and ranking structural types in the appropriate list box, the system checks to see if the user has selected a level of R&D and a method of

manufacture. If the user has not selected a level of R&D and a method of manufacture the system initiates the display of the appropriate dialog message box(s) telling the user that he or she has neglected to select a method of manufacture and/or has not considered the input of R&D. Figure 9.16 shows the Manufacturing and Research and Development Warning dialog message boxes. If either the Manufacturing or Research and Development Warning dialog message boxes are displayed the user should close down the Best Structure dialog box and return to the Research and Development Input and/or the Cost dialog boxes, here selecting the appropriate level of R&D and/or method of manufacture. The code that supports the system checking for manufacturing and R&D input is explained in section C.3.2.2 of appendix C.

Assuming at this juncture, that the user has not neglected to input a method of manufacture, or a level of R&D, and a best structure has been derived, it is possible to view the audit trail that relates to this structure by clicking on the notepad push button immediately below the best structure edit box. The operation of the notepad push button brings up the Best Structure - Audit Trail dialog box, see Figure 9.17. This dialog box presents a list of push buttons that correspond to the seven structural types that the KBS embraces. However, only the push button that corresponds to the structural type which has been defined as the best structure will be enabled. In Figure 9.17, this can be seen to be cellular core. Operating this push button will cause the Cellular Core Audit Trail dialog box to be displayed, see Figure 9.18. The text in this dialog box explains to the user why cellular core is considered to be the best structure; this being in the context of the system responses made by the user leading up to defining the best structure. In addition to text, the Cellular Core Audit Trail dialog box also facilitates the viewing of pictures which show the different forms and uses of cellular core. Figure 9.19 illustrates hexagonal cellular core as used in the Rafale aircraft foreplane.

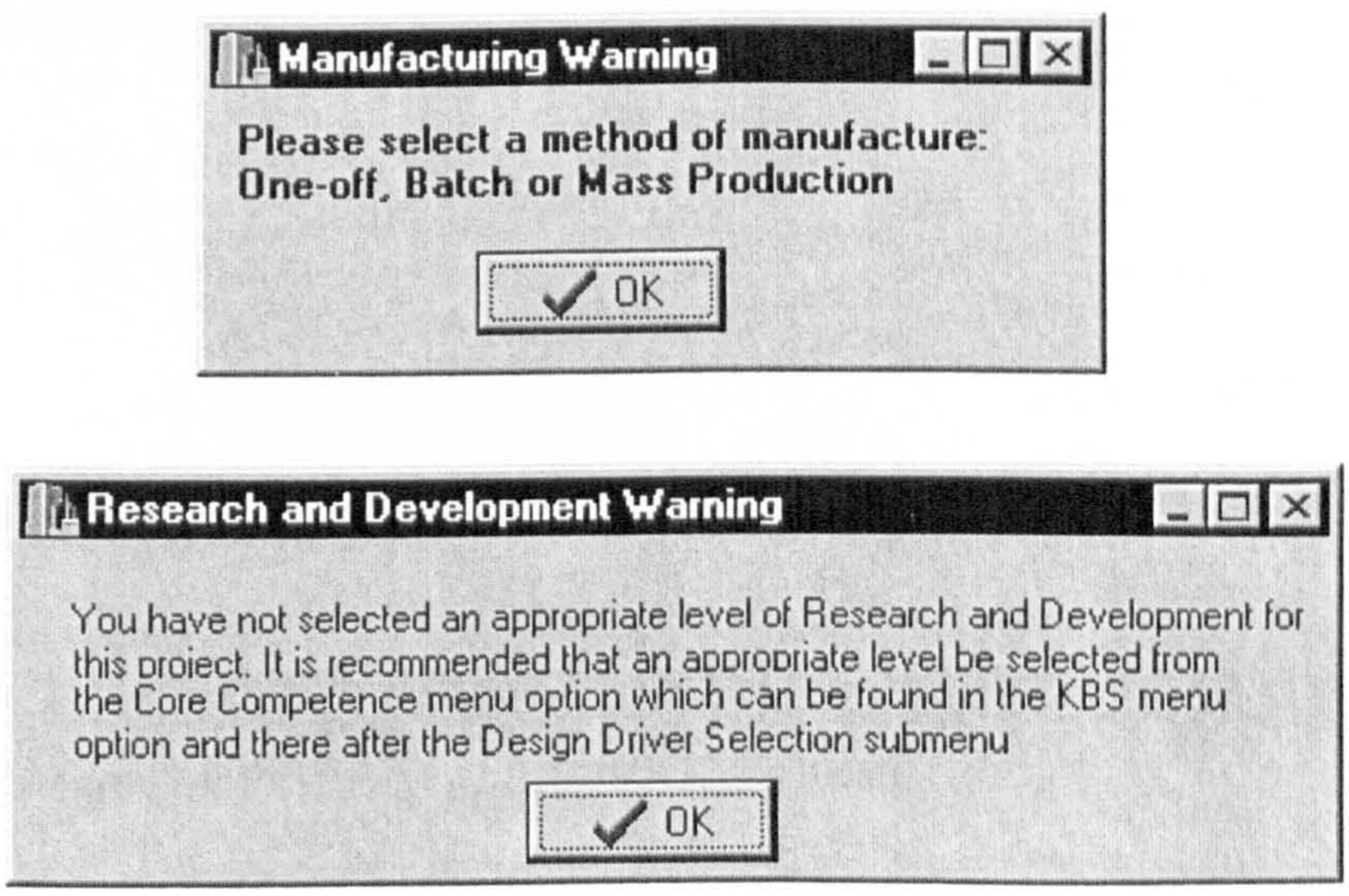



Figure 9.16, The Manufacturing Warning and Research and Development Warning Dialog Message Boxes

In order to facilitate discussion in the context of the impact that the selection of an appropriate level of R&D has on the determination of the best structure i.e., referring to section 9.2.2.2 where it is required that the user select the desired level of R&D in

the Research and Development Input dialog box, it is necessary that it be assumed that a best structure has been declared in the best structure edit box residing in the Best Structure dialog box. With a best structure declared, the system compares this structure with the level of R&D selected. If there is a discrepancy between the selected level of R&D and the declared best structure then this issue is brought to the users attention. For example, if the user selects a minimal level of R&D, and the KBS presents cellular core as the best structure then there is discrepancy as cellular core structures at the time of writing cannot be produced successfully with a minimal level of R&D. The process of codifying the comparison of the best structure with the selected level of R&D is explained in section C.3.2.5 of appendix C. Figure 9.20 shows a Research and Development Warning message dialog box which corresponds to the cellular core structure.

In the situation where the preferred structures of the manufacturing and structures rule bases are not in agreement i.e., the structures at the top of the two list boxes are not the same, it is necessary to access the secondary rules relating to the structure types, as outlined in section 7.2.4.2 of Chapter 7. On the right-hand side of the Best Structure dialog box there are two columns of seven push buttons. Above these two columns there are two headings, 'Manufacturing' and 'Structures' and above these two headings there is the main heading 'Viewpoint'. The push buttons in each column are labelled with the range of structures that the KBS embraces. The user is required to select a preferred viewpoint be it Structures or Manufacturing. The selection will depend on which structures are presented as being the preferred structure by each rule base.

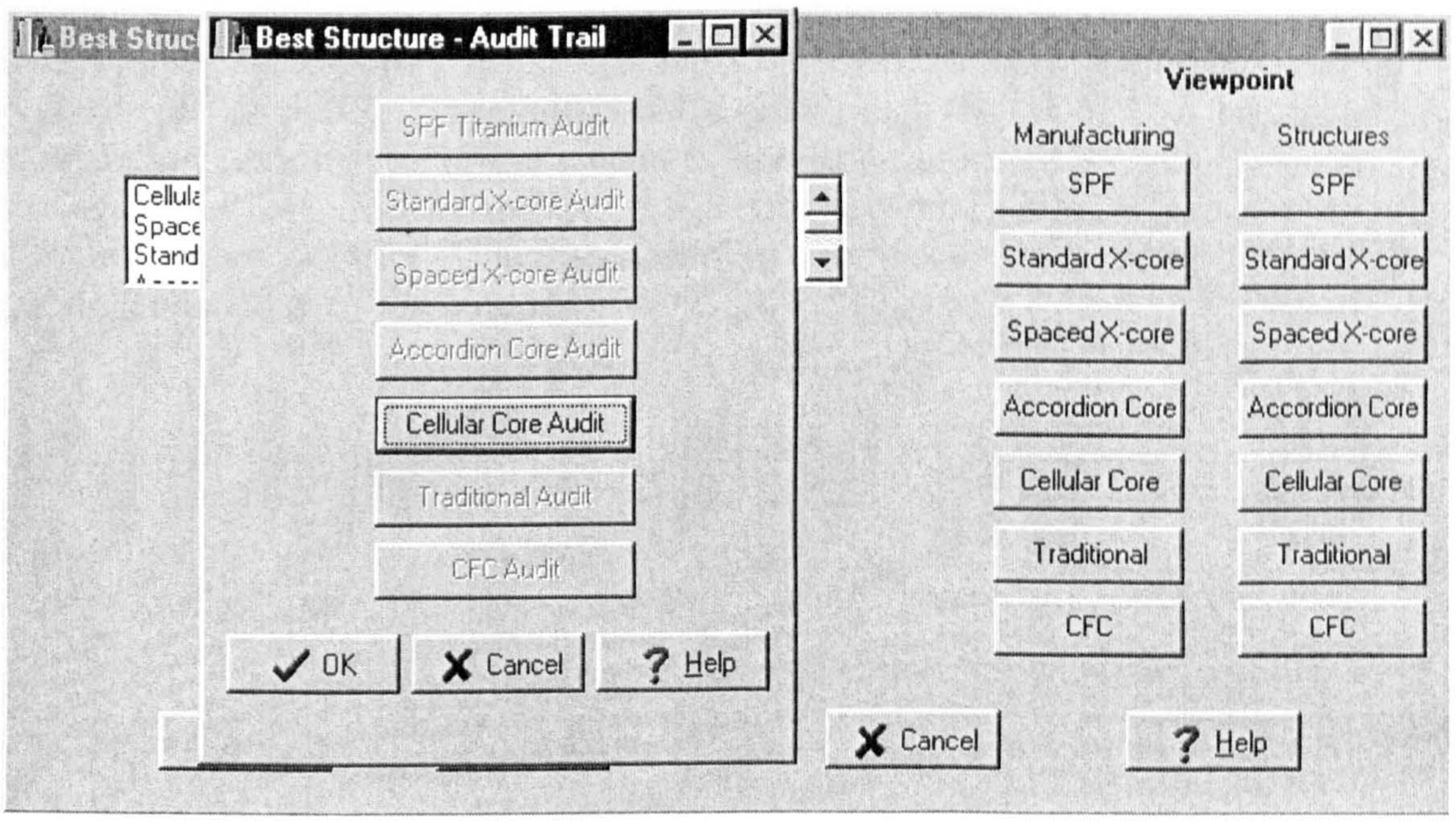



Figure 9.17, The Best Structure - Audit Trail Dialog Box

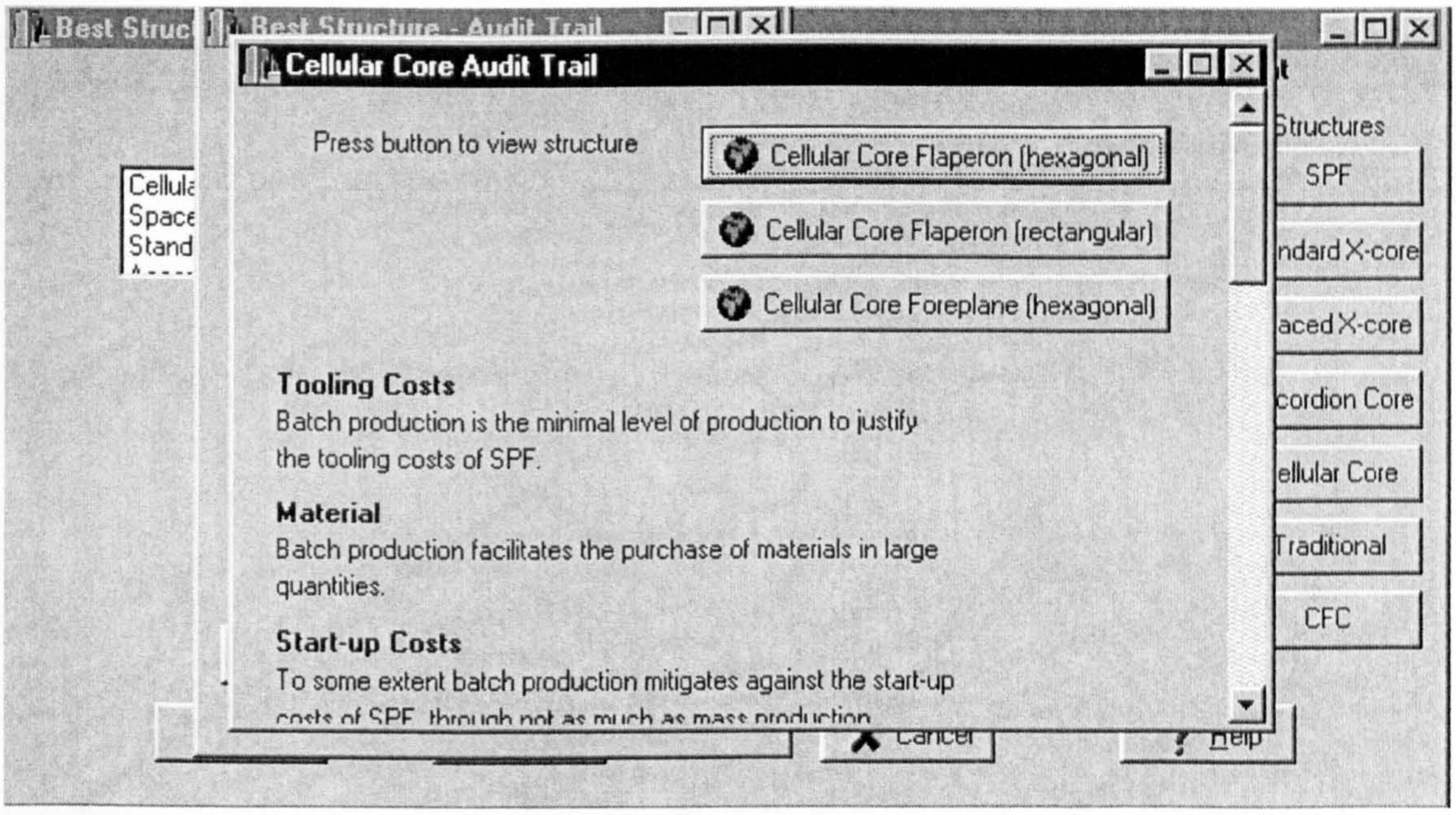



Figure 9.18, The Cellular Core Audit Trail Dialog Box

Assuming that in this instance, cellular core is the preferred choice in the structures rule base and the structural perspective is the preferred perspective. Then the cellular core push button should be selected by the user from the column of seven push buttons below the heading 'Manufacturing'. This operation displays the secondary rules relating to the cellular core structure in manufacturing rule base. These secondary rules are displayed in the Cellular Core Manufacturing Secondary Rules dialog box, see Figure 9.21. The user should scroll through the dialog box and select those secondary rules that he or she wishes to fire. Rules are fired by clicking on the appropriate check box. Having selected the desired secondary rules to fire, the user should close down Cellular Core Manufacturing Secondary Rules dialog box. The user, should now click on the

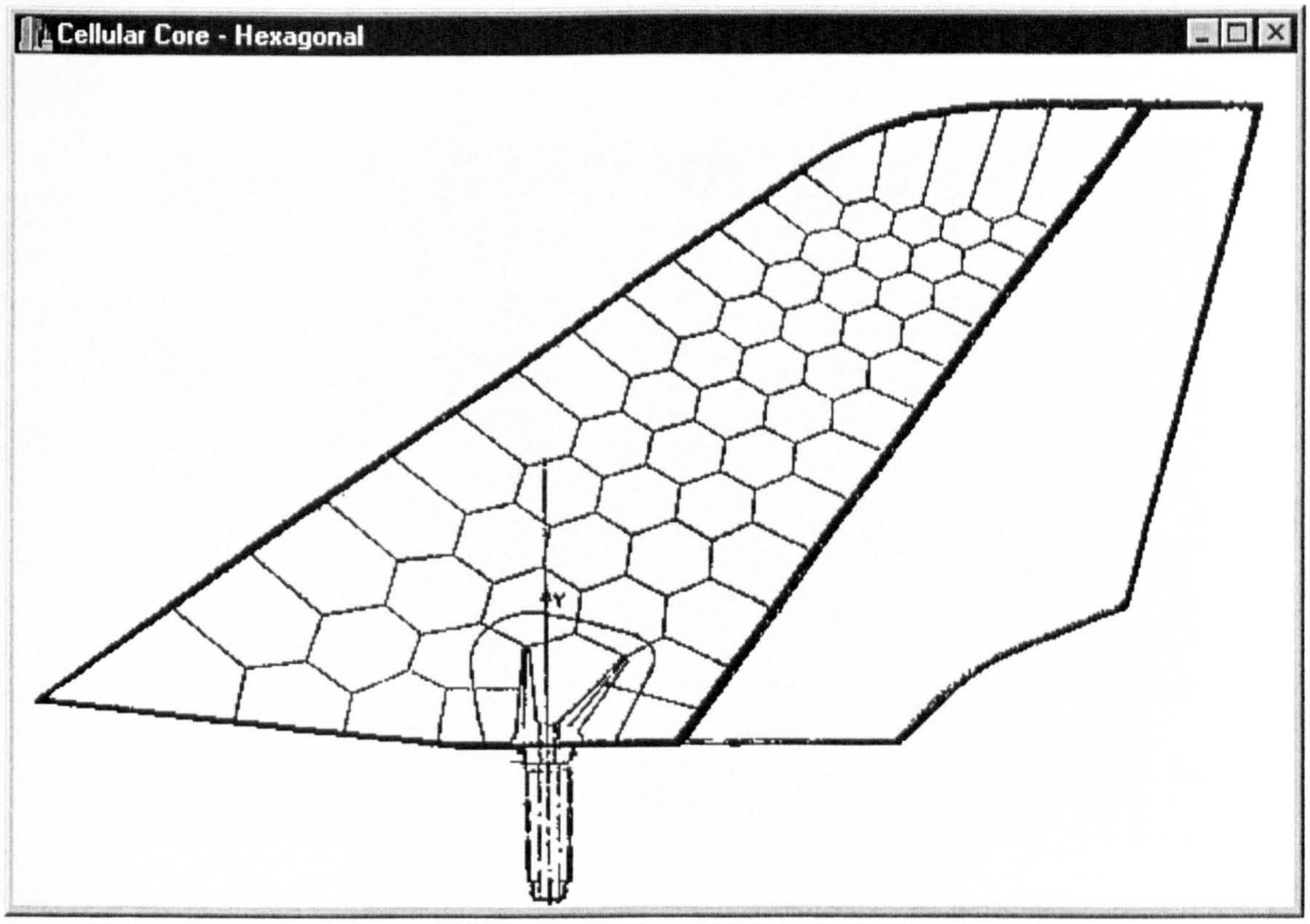



Figure 9.19, Hexagonal Cellular Core as used in the Rafale Aircraft Foreplane

'Best Structure?' push button in the Best Structure dialog box. This causes the secondary rules relating to the cellular core structure to impact on the manufacturing rule base. The secondary rules score in a similar manner to the rules described in section C.3.2.3 of appendix C. Depending on the number of rules fired, the cellular core structure will progressively ascend the list of structures in the list box corresponding to the manufacturing rule base until it becomes the preferred structure for the rule base i.e., it is at the top of the list. Once cellular core becomes the preferred structure for the manufacturing rule base, cellular core will be displayed in the best structure edit box and if the user selects the notepad push button the audit trail relating to cellular core can be accessed as described previously. Section C.3.2.7 of appendix C, describes how the code that supports the operation of secondary rules relating to cellular core in the manufacturing rule base works. It should be noted that the method described for cellular core secondary rules is identical for all other secondary rules supporting other structural types in both the manufacturing and structures rule bases.

Clearly, it may be desirable in the future to expand the manufacturing and structures rule bases by adding new rules. As such, it may be necessary to add to the existing secondary rules. With this in mind, section C.5.8 of appendix C discusses the steps necessary to facilitate the introduction of new secondary rules.

The code underlying the operation of the Best Structure dialog box supports the manufacturing and structures rule bases as described in section 7.2.4 of Chapter 7, which documents the ICES methodology. An important aspect with respect to the

operation of these rule bases is that they embrace seven distinct structural types i.e. super plastic formed (SPF) titanium, standard x-core, spaced x-core, accordion core, cellular core, traditional stressed skin and carbon fibre composite (CFC). In order to

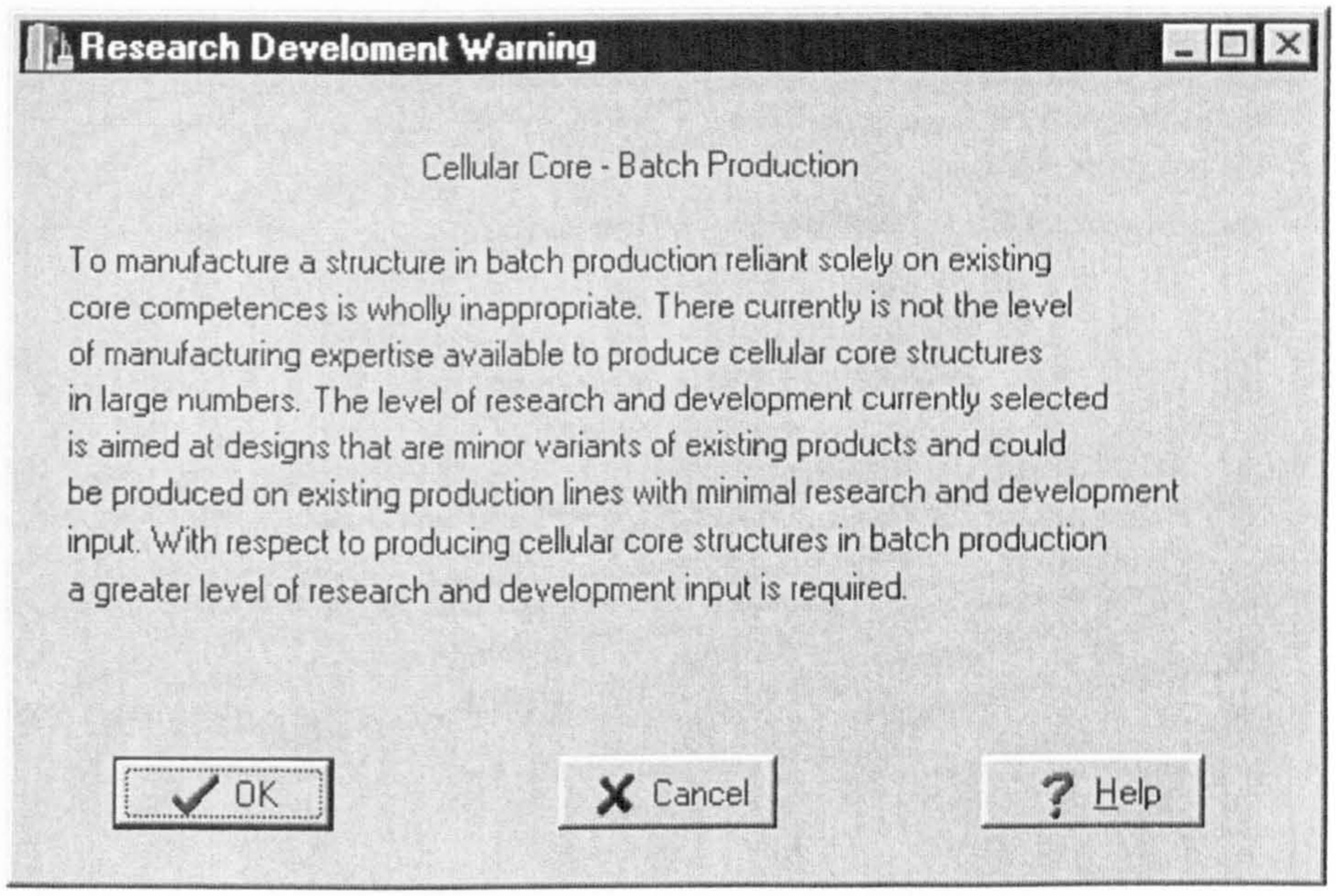



Figure 9.20, Research and Development Warning Dialog Box Corresponding to Cellular Core

ensure there is no inherent bias in the system every rule in both the manufacturing and structures rule bases is applied to every structural type. Clearly, this necessary requirement has the effect of causing a large quantity of code to be generated. This aspect of the prototype is discussed in Chapter 10.

As indicated in section 7.2.4.1 of Chapter 7, all the rules operating in the manufacturing and structures rule bases are able to impact on one or more of seven distinct sections. The reader is reminded that the concept behind defining these sections is that when a rule fires in the manufacturing and/or structures rule bases each rule scores a weighting in one or more of these seven sections. The weighted score for each rule is determined from a positive correlation between the rules residing in the rule bases and the design drivers. In terms of implementing the methodology as described in Chapter 7 in a codified format it was necessary to make every rule within the two rule bases a stand-alone entity. The reader is directed to section C.3.2.3 of Appendix C where the operation of the rules residing in the manufacturing and structures rule bases is explained in detail.

Clearly, it may be desirable in the future to expand the manufacturing and structures rule bases by adding new rules. Section C.5.7 of appendix C discusses the steps necessary to facilitate the introduction of such rules. The ease with which new rules may be entered into the ICES KBS is discussed in Chapter 10.

Turning attention now to the relationship between the ICES KBS and case base. To provide a physical link between the KBS and the case base i.e., facilitate the case base being able to display the cases that contain the best structure, the seven section score totals relating to every structural type in both the structures and manufacturing rule base are added. These numerical scores may then be accessed by the case base and as such used to identify the case(s) which contain the best structure as defined by the KBS. An explanation of this operation and of how these numerical scores are placed in an environment where they can be accessed by the case base is discussed in section C.3.2.6 of appendix C.

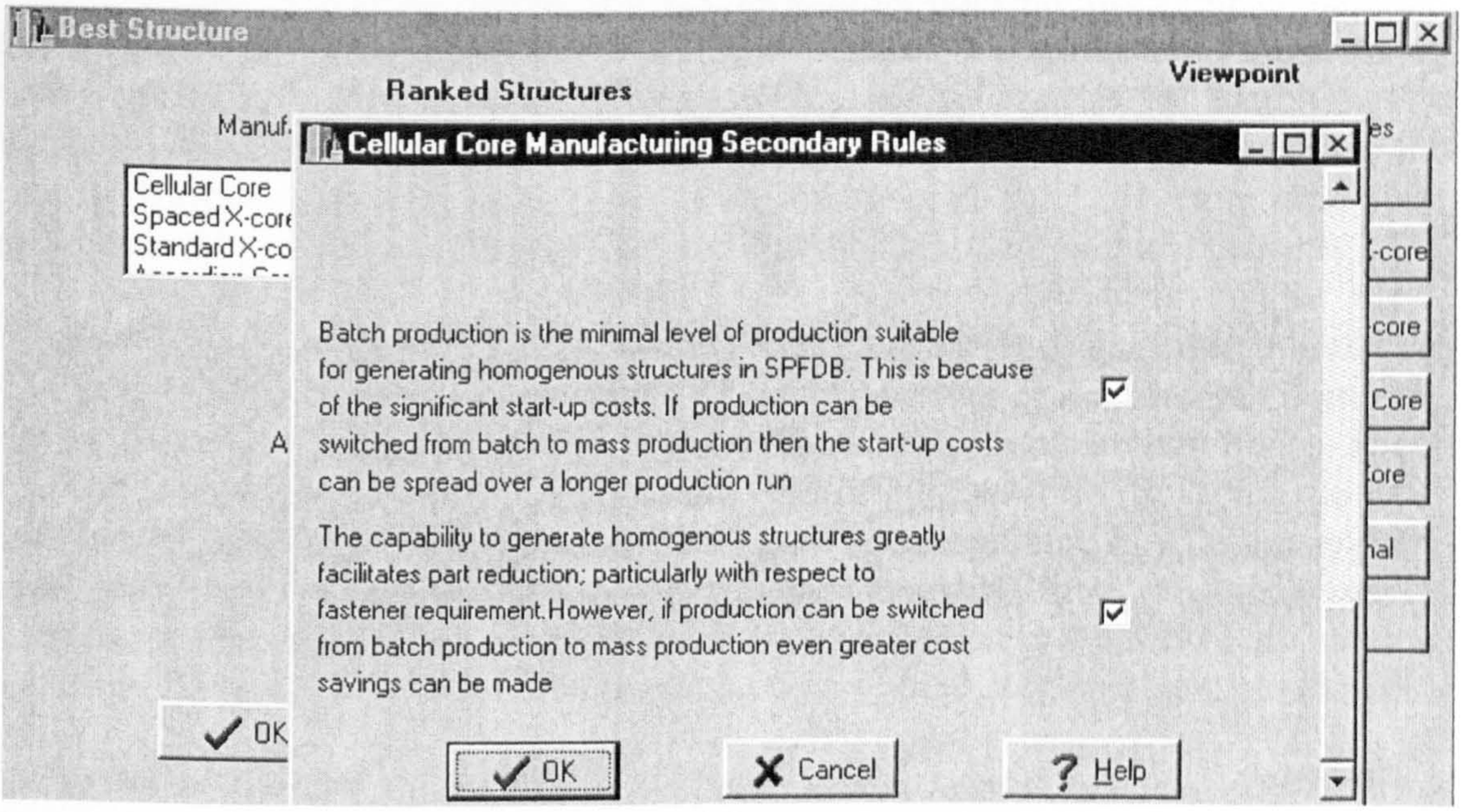



Figure 9.21, Cellular Core Manufacturing Secondary Rules Dialog Box

At this juncture the reader should have a reasonable appreciation of how the ICES KBS derives a best structure and in particular the operation of the Best Structure dialog box. The following section now discusses the case base component of the ICES prototype

### 9.2.3. Case Base Operation

This section will now outline in a logical manner, the operation of the ICES prototype case base capability. Where appropriate reference will be made to the ICES software 'front-end' components and the ICES KBS introduced previously in this chapter. In addition, where appropriate, reference will be made to the ICES methodology as documented in Chapter 7. The code that supports the underlying functionality of the ICES case base is provided in the appropriate sections of appendix C. In addition, the areas of the case base which lend themselves to further development are discussed in section C.5.9 of appendix C; here directions for expanding the underlying code are given. Before proceeding with this description of the operation of the ICES case base, it is important to re-emphasise that an aircraft foreplane was the aircraft structure selected to drive through the ICES prototype in order to validate the underlying methodology as defined in Chapter 7. As such, all the explanations provided in this discussion relate to this aircraft structure. As a final point it should be noted that

occasional reference is made to the Foreplane database table. This is the relational database table created in Paradox which supports the case base. If the reader should become confused at any juncture, the term 'Foreplane database table' may be read as case base. Through not accurate it will suffice for explanation purposes.

### 9.2.3.1. Entering a New Case

When the Enter Case menu option is selected from the Case Base menu the New Case window is presented to the user. The New Case window is illustrated in Figure 9.22. As commented upon in section 9.2.1.5.1, C++ Builder does not discriminate between dialog boxes and windows. However, in the context of this application, those dialog boxes which have their own menuing capability are referred to as windows. The purpose of the New Case window is to enable new design cases to be entered into the system. Before discussing how new cases are entered into the case base it is appropriate first to discuss the user interface that the New Case window presents.

It can be seen from Figure 9.22, that the New Case window apart from possessing a menu also has a large number of fields displaying a range of information. Down the left-hand side of the figure it can be seen that there is a column of edit boxes. These edit boxes identify the design case number, the component title, the structure type, followed by a range of material and performance specifications. In addition to these edit boxes, the New Case window has five memo boxes and three graphics boxes. If these memo and graphic boxes can not be viewed immediately the New Case window is initiated they can be brought into view by use of vertical and horizontal scroll bars.

Focusing attention on the five memo boxes. These memo boxes provide detailed descriptive information relating to the foreplane design cases residing in the case base. It should be noted that it is possible to page-up and page-down the memo box fields i.e., not all the information available within the memo boxes is visible. The specific subject areas of the foreplane design cases covered in these memo boxes is as follows:

- Component Description
- Structure Description
- Manufacturing
- Defence
- Risk

The Component Description memo field provides a detailed description of the component of interest i.e., in the context of this study a foreplane. In particular, the description focuses on the generic function of the component rather than specific details relating to the particular design case. The Structure Description memo field provides a detailed description of the structure used within the design case e.g. carbon fibre composite (CFC), cellular core, spaced x-core etc. The Manufacturing memo field discusses the manufacturing processes involved with respect to the foreplane design case in question. The Defence memo field presents a defence of the foreplane case presented to the user; the purpose of this field is to present positive arguments for the case. This field has a key role to play in applying the jury technique as discussed in section 9.2.3.2.2. The Risk memo field presents the risks involved in selecting the foreplane case presented. This field also has an important role to play in applying the jury technique as will be apparent in section 9.2.3.2.2 when the jury technique is

discussed. Looking now at the three graphic fields residing in the New Case window. The three graphic fields are:

- Aircraft Type
- Engineering Drawing
- Structure Graphic

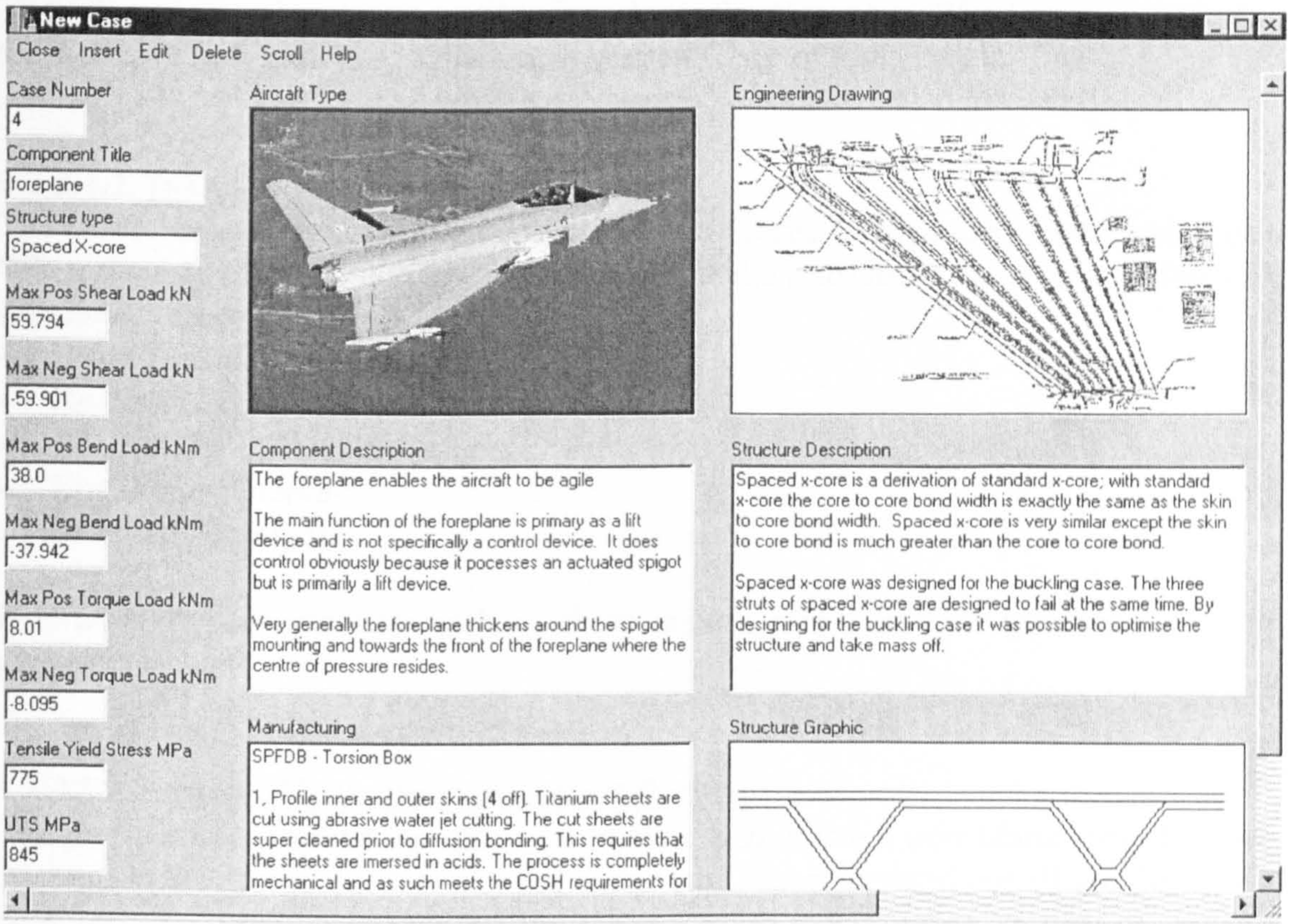



Figure 9.22, The New Case Window

The Aircraft Type graphic field presents a picture of the aircraft from which the foreplane in the design case in question comes from. In generic terms, the purpose of this picture is provide the user with an appreciation of the orientation of the component represented within the design case. The Engineering Drawing graphic field provides a plan view engineering drawing. The aim of which is to provide the user with an appreciation of the structural configuration of the foreplane presented in the case. The purpose of the Structure Graphic graphic field is to provide a pictorial representation of the structure described in the Structure Description memo field, and hence enhance the user's appreciation of the salient points of the structure presented in any particular foreplane case.

Having introduced all the fields residing in the New Case window it is now appropriate to turn attention to the menu. The purpose of the New Case window menu is to provide the functionality necessary to facilitate the entry of new cases into the case base and enable cases residing in the case base to be edited. From Figure 9.22 it can be seen that there are six main menu items in the New Case window i.e., Close, Insert, Edit, Delete, Scroll and Help. These menu items support pull-down menus which provide the functionality necessary to enter and edit design cases. It is now

proposed to discuss each of these menu items in turn, and in so doing the reader should gain a clear understanding of the operation of the New Case window.

#### 9.2.3.1.1. Close

The Close menu permits the user to close down the New Case Window. The code that facilitates this operation is explained in section C.4.1 of appendix C.

#### 9.2.3.1.2. Insert

The Insert menu command enables new design cases to be added to the case base. When the user clicks on the Insert menu command a pull-down menu is revealed with five menu options:

- Insert Record
- Re-Number Records
- Aircraft Type Graphic
- Eng. Draw. Graphics
- Structure Graphics

It is now proposed to take each of these menu options in turn and describe their purpose.

##### 9.2.3.1.2.1. Insert Menu Options - Insert Record

The purpose of the Insert Record menu option is to create a new blank record. When this option is selected the user will see a new case record created but all the fields will be empty. The code that supports the Insert Record menu option i.e., facilitating the generation of a new case record, is explained in section C.4.2.1 of appendix C.

##### 9.2.3.1.2.2. Insert Menu Options - Re-Number Records

The purpose of the Re-Number Records menu option is to provide the user with the means of re-numbering design case records residing in the case base independently of any other operation. In the instance where a new design record is inserted in mid-table it is desirable that all the design case records in the case base be re-numbered from the first design case record and not from the insertion point. The Re-Number Records menu option facilitates this capability. The code that supports this re-numbering operation is discussed in section C.4.2.1 of appendix C.

##### 9.2.3.1.2.3. Insert menu options - Aircraft Type Graphics, Eng. Draw. Graphics, Structure Graphic

It is appropriate to discuss the remaining three menu options in the Insert menu together i.e., Aircraft Type Graphics, Eng. Draw. Graphics and Structure Graphic, as their purpose is very similar. Once a new empty design case record as been entered into the case base these menu options enable the user to enter graphics into the appropriate fields of the record. The code that supports the assigning of graphics to a new design case record is explained in section C.4.2.2 of appendix C.

#### 9.2.3.1.3. Edit

The Edit menu command enables the user to edit the fields of design cases residing in the case base. When the user clicks on the Edit menu command a pull-down menu is revealed which has a menu option corresponding to every field in the New Case window, see Figure 9.23. It should be noted that by default all cases in the case base are read only. Thus, to edit cases the user has to deliberately select the appropriate case field from the Edit menu. Clearly, there are times when it is desirable to be able to edit all the case fields at once. With this in mind, the first menu option in the Edit menu

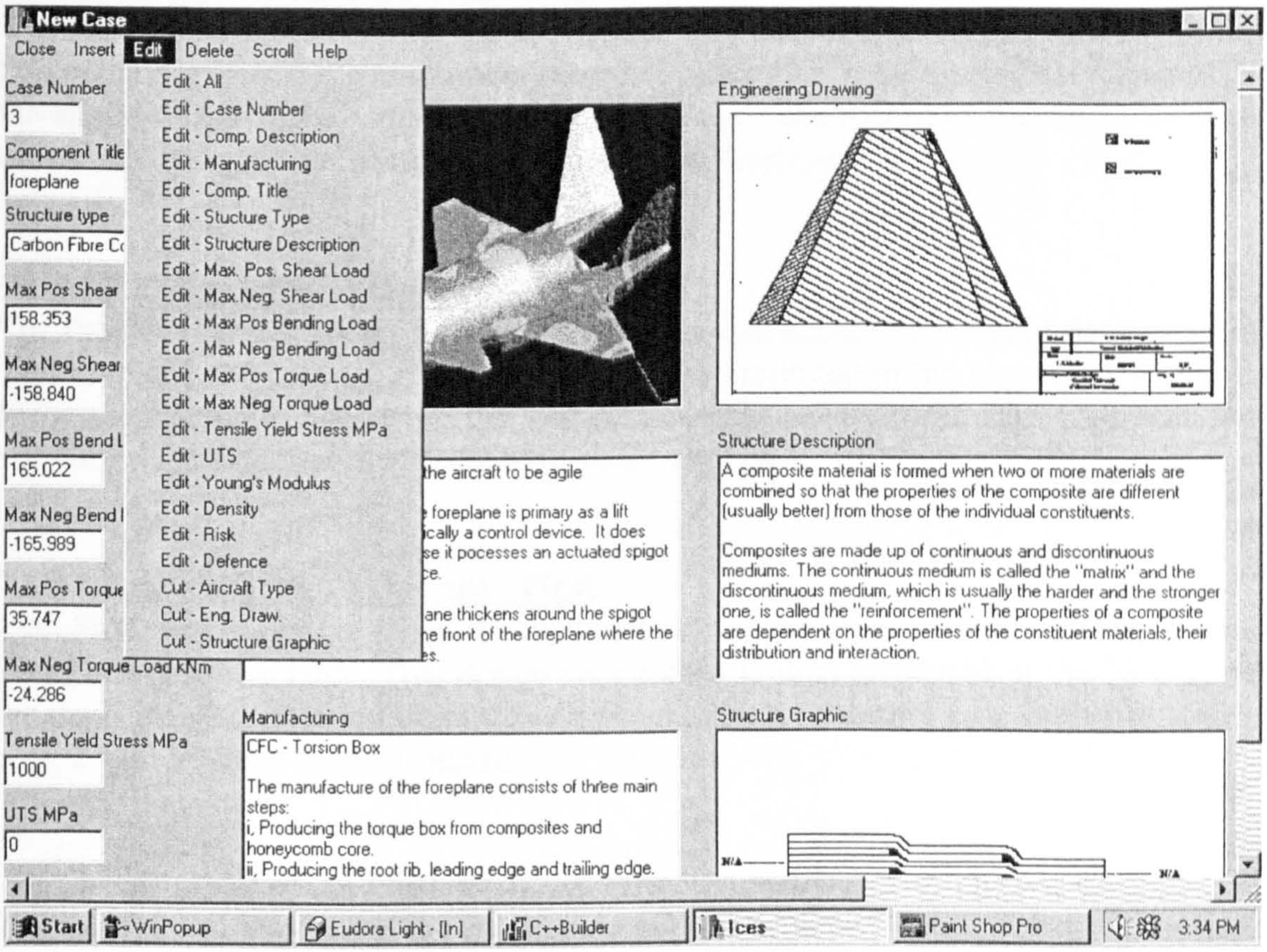



Figure 9.23, New Case Window Displaying Edit Menu Options

is Edit - All. This menu option enables any field of a case record to be edited. In addition to enabling the fields that support textual and numerical information to be edited the Edit menu also provides three menu options that allow the user to remove the graphics from any particular design case record. These menu options are Cut - Aircraft Type, Cut - Eng. Draw. and Cut - Structure Graphic, see Figure 9.23. The code that supports the editing of design case records is explained in section C.4.3 of appendix C.

#### 9.2.3.1.4. Delete

The Delete menu has one pull-down menu option called Delete Record. As its name implies, this menu options enables the user to delete any design case record residing in the case base. An explanation of the code that supports this menu option is provided in section C.4.4 of appendix C.

#### 9.2.3.1.5. Scroll

The purpose of the Scroll menu option is to enable the user to scroll through the design case records residing in the case base and access the one required with relative ease. The Scroll menu option has four pull-down menu options i.e., Next, Previous, First and Last. The code that supports these menu options is provided in section C.4.5 of appendix C. It is now proposed to briefly discuss each of these menu options.

##### 9.2.3.1.5.1. Scroll Menu Options - Next

The Next menu option scrolls the application one design case record forward in the Foreplane database table and a new record is presented in the New Case window. If the application is already displaying the last design case record in the case base then the current design case record will continue to be displayed.

##### 9.2.3.1.5.2. Scroll Menu Options - Previous

The Previous menu option scrolls the application one design case record backwards in the Foreplane database table and a new record is presented in the New Case window. If the application is displaying the first design case record in the case base then the current design case record will continue to be displayed.

##### 9.2.3.1.5.3. Scroll Menu Options - First

The First menu option takes the application to the first record residing in the Foreplane database table. This design case record is then presented in the New Case window. If the application is already displaying the first design case record then this record will continue to be displayed.

##### 9.2.3.1.5.4. Scroll Menu Options - Last

The Last menu option takes the application to the last record residing in the Foreplane database table. This design case record is then presented in the New Case window. If the application is already displaying the last design case record then this record will continue to be displayed.

#### 9.2.3.1.6. Help

When the Help menu option is selected the application presents the user with the Case Base Help dialog box. This dialog box outlines the operation of the New Case window, focusing particularly on the menu functionality as described in section 9.2.3.1.1 onwards.

### 9.2.3.2. Querying the Case Base

When the Case Query and Jury menu option is selected from the Case Base menu the Case Query and Jury window is presented to the user. The Case Query and Jury window is illustrated in Figure 9.24. The purpose of the Case Query and Jury window is to enable the case base to be queried. The user can query the case base via the Case

Query and Jury window in two distinct ways. Firstly, the user can identify those design cases which possess the best structure as defined by the KBS. Secondly, through the entry of known specifications the user can identify the best case residing in the case base whose specifications correspond most closely with those entered. The identification of the best case in this manner is supported by the application of a jury technique. The methodology supporting the jury technique is discussed in section 7.3.3 of Chapter 7. In addition to being able to query the case base, the Case Query and Jury window provides the means by which the user can also adapt cases. The methodology that under-pins case adaptation as applied in the ICES prototype is discussed in section 7.3.4 of Chapter 7.

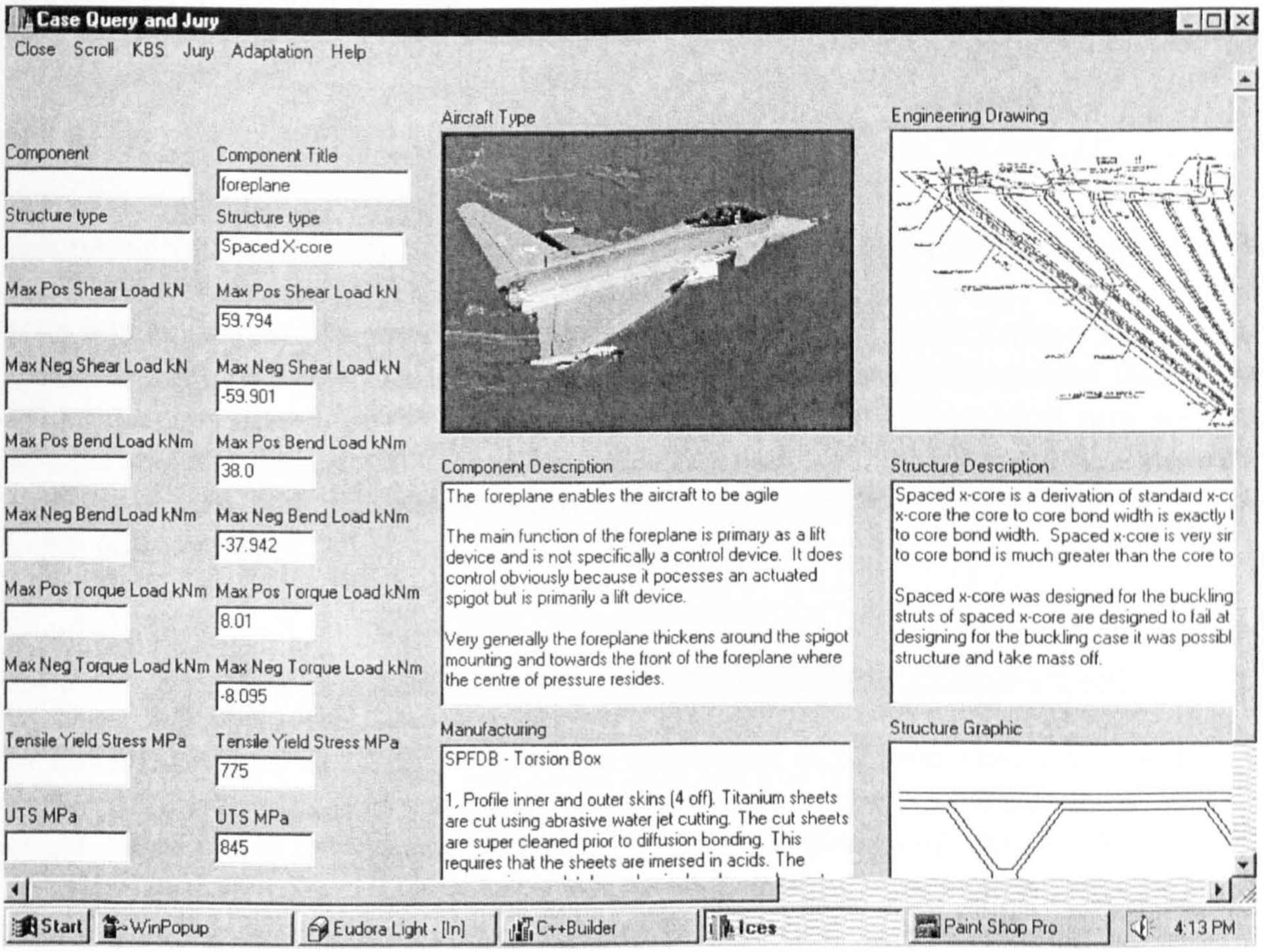



Figure 9.24, The Case Query and Jury Window

Before discussing the two methods of querying the case base and case adaptation, it appropriate to first discuss the user interface presented by the Case Query and Jury window. It can be seen from Figure 9.24 that the Case Query and Jury window is very similar to the New Case window. In addition to containing all the design case record fields commented upon when discussing the New Case window in section 9.2.3.1 there is an additional memo box and several new edit boxes. The memo box supports a new field titled Adaptation Notes. This field will be discussed in the context of case adaptation in section 9.2.3.2.3. As can be seen from Figure 9.24, the new edit boxes are listed down the left-hand side of the Case Query and Jury window. The purpose of the these edit boxes is to enable the user to enter known specifications and then in conjunction with the jury technique identify the best case residing in the case base. The use of these edit boxes in the context of querying the case base will be discussed in section 9.2.3.2.2.

In common with the New Case window, the Case Query and Jury window has a menu. The Close and Scroll menu options in the Case Query and Jury window are identical to the same menu options in the New Case window as discussed in section 9.2.3.1.1 and section 9.2.3.1.5 respectively. The reader who requires further information relating to these menu options should refer to these sections. The Case Query and Jury window also has a Help menu option. When the Help menu is selected the user is presented with the Case Base Query Help dialog box. This dialog box explains how the case base may be queried via the Case Query and Jury window. The remaining menu options presented in the Case Query and Jury window relate to the methods of querying and adapting design cases residing in the case base i.e., KBS, Jury and Adaptation. These menu options will be discussed in the appropriate subsequent sections.

#### 9.2.3.2.1. Selection of the Best Case via the KBS

Assuming that the user has identified a 'best structure' through the KBS component of the ICES software prototype, it is now possible to identify the case(s) within the case base which contain this structure. The user should select the KBS menu in the Case Query and Jury window. This menu has one pull-down menu option called Best Case. When the user selects this Best Case menu option, the Case Query and Jury window will present the first foreplane case from the case base that possesses the best structure as identified by the KBS. The code that supports this operation is explained in section C.4.6 of appendix C.

#### 9.2.3.2.2. Selection of the Best Case via Nearest Neighbour Matching and Applying the Jury Technique

In the introduction to Chapter 7, it was stated that ICES was developed with two modes of operation in mind, and two possible scenarios where outlined where the user could interact with the system. In the second scenario discussed, the user has a set of specifications, from which he or she would wish to know whether any design cases residing in the case base represent a close match with the specifications. The user enters the specifications into the system. The system then through nearest neighbour matching compares the specifications to those of the design cases in the case base and presents the user with the best match.

As discussed in section 7.3.3 of Chapter 7, there are limitations of such a method of interrogating the case base due largely to the potentially huge size of individual aircraft cases. The problem being that the user will very unlikely have a complete set of specifications which will embrace all aspects of the design cases residing in the case base. Thus, it is likely that several aspects of the best matching case will be presented without justification. As indicated in section 7.3.3, to overcome this problem the jury technique was developed. In conjuncture with the nearest neighbour matching technique as discussed in section 7.3.2, the jury technique represents a novel and effective method of deriving the best design case in the case base. It is now proposed to discuss how these techniques were implemented within the ICES case base.

As indicated in section 9.2.3.2, the Case Query and Jury window has a series of edit boxes listed down its left-hand side. These edit boxes permit the user to enter a range of known material and performance specifications, see Figure 9.24. Once all the specifications have been entered into these edit boxes the user should select the Best Case menu option from the Jury menu. The operation of the Best Case menu option has the effect of deriving the best case through nearest neighbour matching. The code that supports the nearest neighbour matching technique, and the jury technique, which is discussed in the following paragraphs, is explained in section C.4.7 of appendix C.

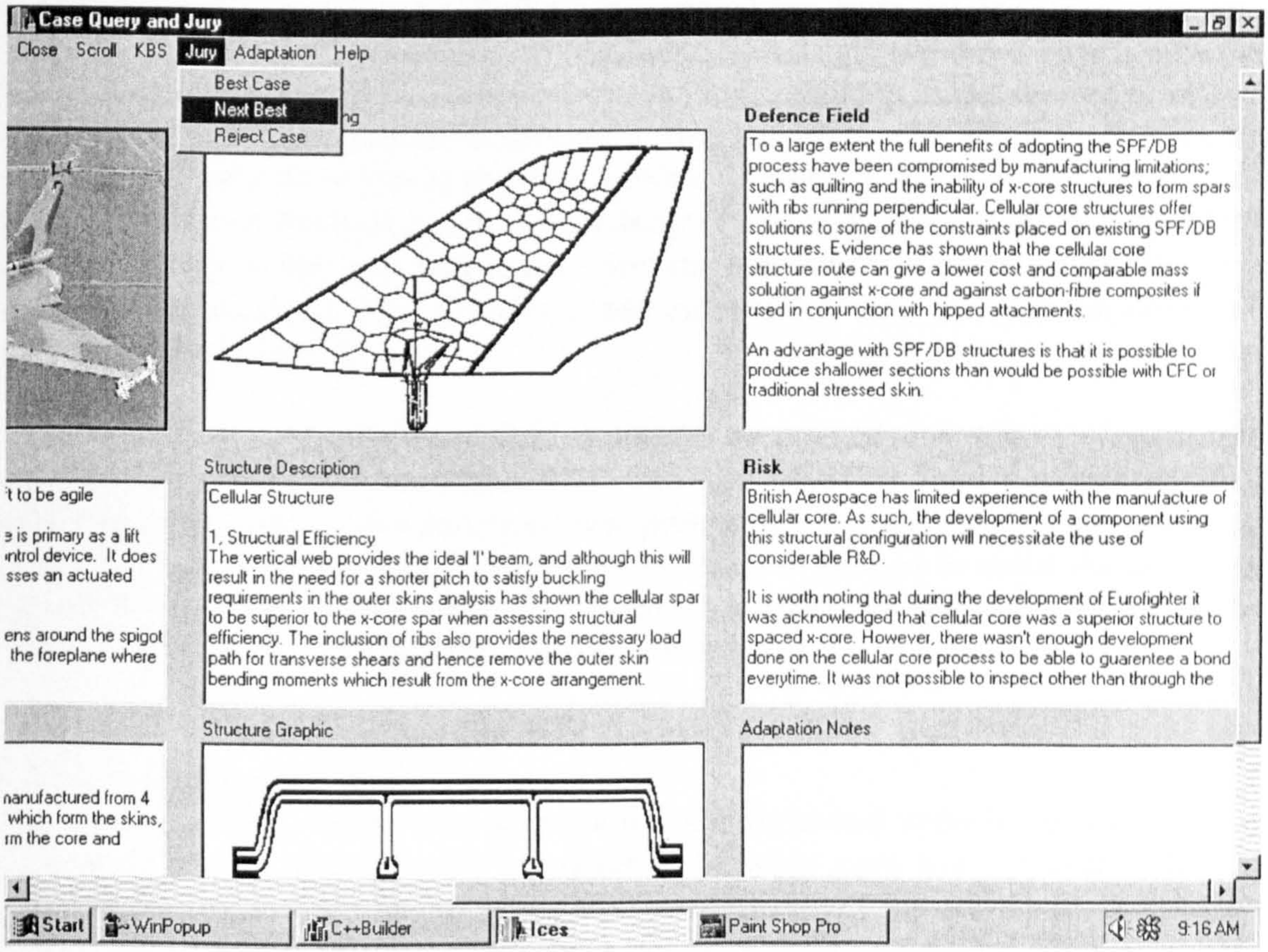



Figure 9.25, The Case Query and Jury Window Displaying the Defence Field

With the best case as defined by the nearest neighbour technique presented to the user in the Case Query and Jury window, the jury technique is now applied. As discussed in section 7.3.3 of Chapter 7, when the system presents the case that is the best match to the input specifications ICES requires that this case defend itself with respect to the next best case in the case base. The method by which a case defends itself is through the Defence field as illustrated in Figure 9.25. In this field the case states the reasons why this particular case is better than alternative cases available in the case base.

The user through selecting the Next Best menu option from the Jury menu will cause the application to display the next best case in the context of the nearest neighbour matching technique. With the next best case displayed in the Case Query and Jury window the user can read the defence that this case presents in it's Defence field. It is possible for the user to re-examine the best case by selecting the Best Case menu option from the Jury menu. Thus, the user can move the application to and from

the best case to the next best case as desired, comparing the defence that each design case puts forward. If the user decides that the best design case is indeed the best case then no further user input is required and this case may be the used as the basis for solving the current design problem. However, if it is considered that the defence put up by the next best case is superior to the best case then the best case may be rejected. To reject the best case the user should select the Reject Case option from the Jury menu.

If the user now selects the Best Case menu option from the Jury menu the next best case will now be displayed in the Case Query and Jury window. This is now the best case as defined by nearest neighbour matching technique. If the user now follows this by selecting the Next Best menu option from the Jury menu a new next best case is displayed in the Case Query and Jury window. As before, the user can iterate back and forth between the best and next best design cases and compare the defences they present. If desired, the user may again reject the best case if it is considered the next best case is preferable. This procedure may be repeated until the user considers the best case to be indeed the best.

In addition to the Defence field it should be pointed out that all design case records have a Risk field, see Figure 9.25. The Risk field highlights the risks involved with respect to selecting the particular case presented e.g., gaps in knowledge relating to the manufacturing process. The purpose of the Risk field is to assist the user with respect to deciding whether one case is superior to another i.e., there is a balance to be struck by the positive and negative aspects of a case.

#### 9.2.3.2.3. Case Adaptation

The ICES case base offers case adaptation. As commented upon in section 7.3.4 of Chapter 7, in a situation where an aspect of a design case has not been defended successfully it possible for the equivalent component in the next best case to be substituted. It should be noted that the adaptation capability of the ICES case base enables any part of any design case to be substituted for any other. The adaptation capability of the ICES case base is accessed via the Adaptation menu as illustrated in Figure 9.26. It is now proposed to discuss in the remainder of this section how a design case may be adapted.

The first step necessary to create an adapted case is to create the platform on which the adapted case is to be built. What is required here is to generate a blank design case record and then build the adapted case on this. A blank design case record is created when the user selects the Create Adaptation Platform from the Adaptation menu. As discussed in section 7.3.4, there is a need to identify adapted cases as being different from real aircraft cases. This is because they are not as creditable as fully validated cases e.g., the European Fighter Aircraft (EFA) or the Experimental Aircraft Programme (EAP). As such, when a blank adaptation record is created it is bookmarked as being an adapted case in order to distinguish it from real aircraft cases.

With the blank adaptation platform created, the user should copy an existing real case onto this platform. The user should either select the best design case record or scroll through the cases residing in the case base until the desired one is displayed. It

can be seen from Figure 9.26 that the Case Query and Jury window has a scroll menu option. This operates in an identical manner to the scroll capability in the New Case window as described in section 9.2.3.1.5. With the desired case displayed, the user should select the menu option Bookmark Case from the Adaptation menu. When the desired case has been selected and a bookmark assigned, the user should select the Copy Record menu option from the Adaptation menu. This menu option will cause the selected case to be copied to the blank adaptation platform.

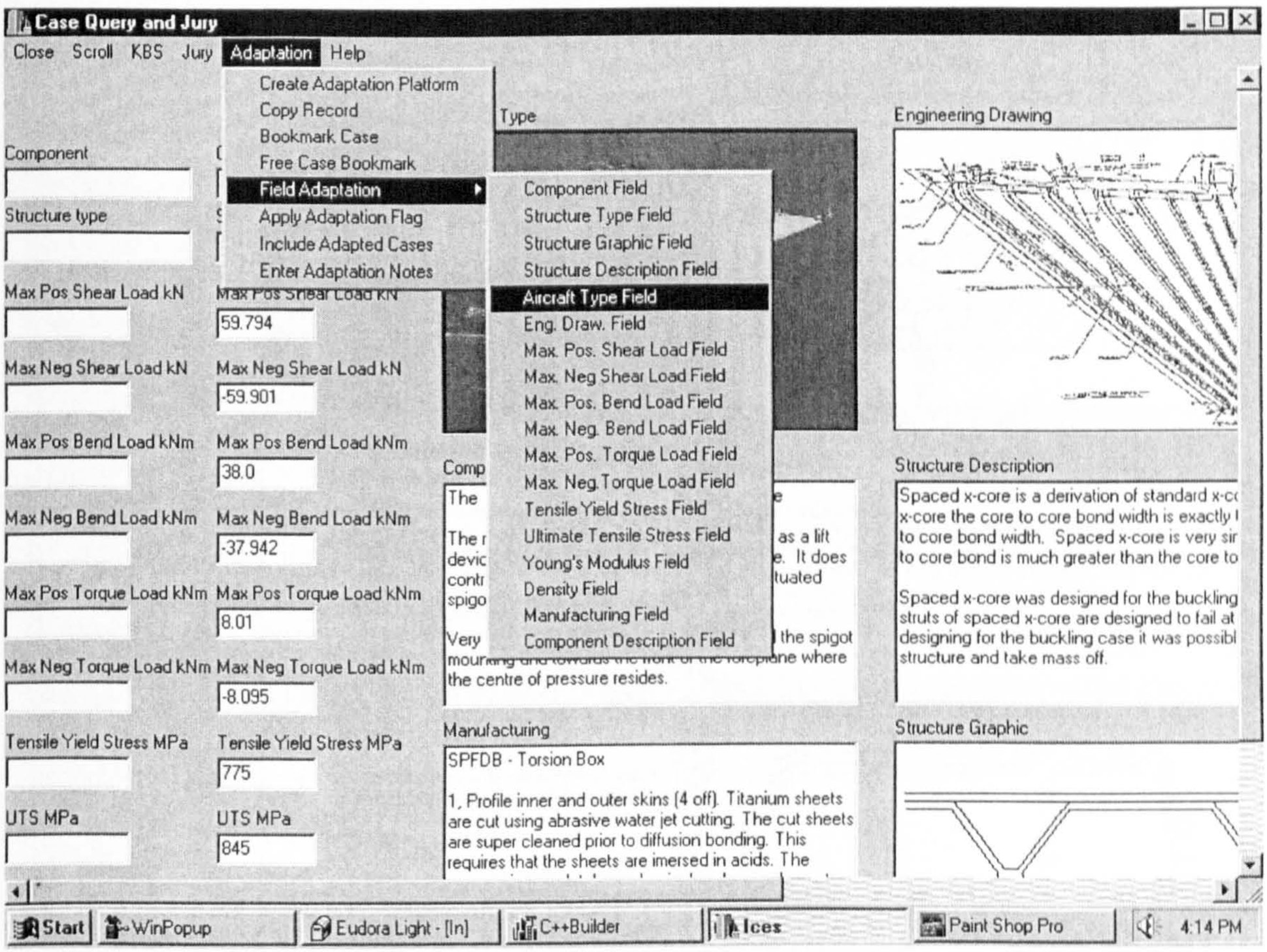



Figure 9.26, The Case Query and Jury Window Displaying the Adaptation Menu Options

With an existing real case copied to the adaptation platform, the user should now select the Free Case Bookmark menu option from the Adaptation menu. Having freed the bookmark, the user should scroll the application to the desired design case record which contains the aspect which it is intended to substitute in the equivalent field in case presented in the adaptation platform. With the appropriate case record displayed, the user should again select the Bookmark Case menu option from the Adaptation menu. Following the bookmarking of the selected case record the user should select the Field Adaptation menu option from the Adaptation menu. As can be seen from Figure 9.26, the Field Adaptation menu provides access to a further sub-menu. This sub-menu enables the user to select the desired field of the displayed case record that it is required to copy to the adaptation platform. After the appropriate sub-menu option(s) from the Field Adaptation menu option have been selected the user may scroll back to the adaptation platform where the field(s) selected by the user can be seen to have been substituted into the appropriate fields of the adapted case.

Figure 9.27 provides an example of case adaptation. It can be seen from this figure that the structure cellular core has been substituted for spaced x-core in the design case corresponding to the EFA foreplane. Adapted case records provide an additional field i.e., Adaptation Notes, where the user may enter notes relating to reasons for adapting the case in the manner chosen. Notes may be entered in the Adaptation Notes field by first selecting the Enter Adaptation Notes menu option from the Adaptation menu.

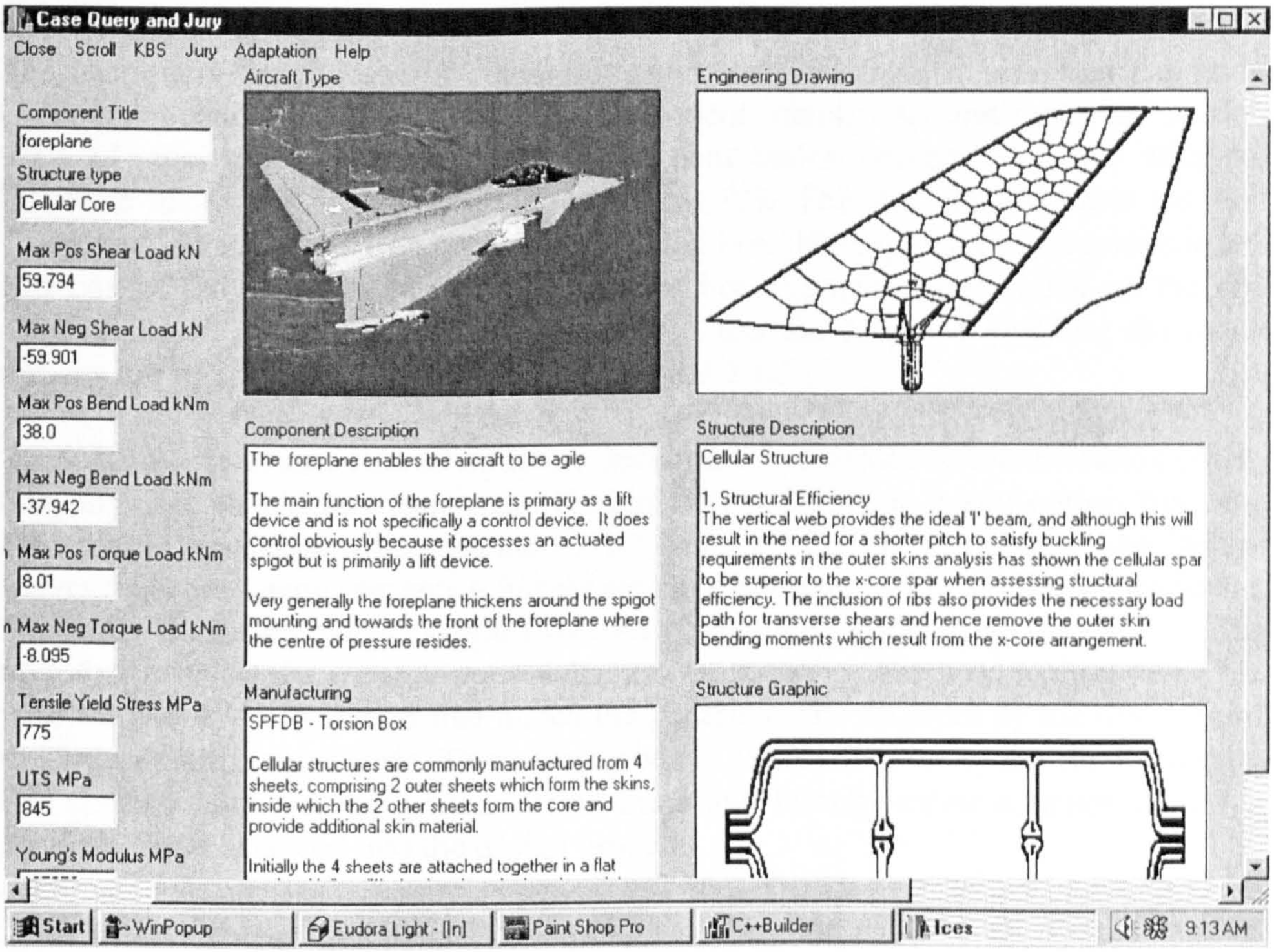



Figure 9.27, An Example of Case Adaptation

By default real cases are kept separate from adapted cases, as adapted cases are not normally included in queries of the case base. However, if the user wishes to query all cases in the case base the Adaptation menu provides the menu option Include Adapted Cases. This menu option causes all design cases in the case base to be treated the same. The user may also reverse this operation and exclude adapted cases from the query process by selecting the Apply Adaptation Flag menu option from the Adaptation menu. The code that supports the ICES case base adaptation capability is explained in section C.4.8 of appendix C.

## 9.3. ICES Prototype Results

This final section of this chapter documents the results output by the ICES prototype. This section starts with the ICES KBS, identifying the best structure through the entry of appropriate user inputs. Having identified the best structure via the ICES KBS, attention will be turned to the ICES case base component. Here the case base is queried firstly, in the context of identifying the best design case with respect to the

best structure identified by the KBS and secondly, via the entry of user input specifications and the application of the jury technique. During this discussion and subsequent presentation of the ICES prototype results, all user inputs and system outputs will be presented in a logical manner. Before proceeding, the reader is again reminded that an aircraft foreplane was the aircraft structure it was decided to drive through the system. As such, all inputs and subsequent outputs relate to this structure.

### 9.3.1. ICES KBS Results

The user starts the process of determining the best structure by selecting the Enter Component menu option from the Component menu. As indicated in section 9.2.1.2.1, the selection of the Enter Component option presents the user with the Structure Identification dialog box, see Figure 9.3. The user should enter the text 'foreplane' in the edit box residing in this dialog box. Having entered foreplane in the edit box of the Structure Identification dialog box the user should click on the OK push button in this dialog box. This operation has the effect of enabling the menu options in the KBS menu as outlined in section 9.2.1.3.

With the foreplane aircraft structure declared in the Structure Identification dialog box, the user should now select the Design Driver Selection menu option from the KBS menu. As discussed in sections 9.2.1.3.1 and 9.2.2.2, the selection of the Design Driver Selection menu presents a further series of sub-menus i.e., Agility, Supersonic, Subsonic, Core-Competences, and Cost. As commented upon in section 9.2.2.2, the user should select these menu options in turn. In so doing, the user will be presented with a series of dialog boxes that match the generic terms declared by the menu items to a set of generic meta rules. These dialog boxes permit the user to fire the meta rules that corresponding to the generic menu terms and thereby cause a range of design drivers to be introduced into the design process.

The user should first select the Agility menu option. This menu option presents the Agility dialog box. The Agility dialog box is illustrated in Figure 9.11. It can be seen from this figure that there are four check boxes corresponding to four design drivers, adjacent to which there are four notepad push buttons. The selection of notepad push buttons on the right-hand side of this dialog box cause dialog message boxes to be presented to the user. Each of these dialog message boxes tell the user the meta rule that will be fired if the user clicks on the corresponding check box. As indicated in section 9.2.2.2 it is the user who decides which design drivers to introduce into the design process.

Taking each of the meta rules residing in the Agility dialog box in turn. The meta rule presented by the dialog message box corresponding to minimum weight design driver states that "To achieve agility there is a requirement for a good instantaneous turn rate and a good sustained turn rate. Both require low wing loading. Wing loading is minimised by reducing weight". The user must now decide whether or not to click on the check box corresponding to this rule and introduce the minimum weight design driver into the design process. Turning attention now to the high structural strength design driver declared in the Agility dialog box. The meta rule presented in the dialog message box corresponding to this design driver states that "To achieve agility there is a requirement for good beyond visual range turns. This requires

a minimum turn radius possibly performed at supersonic speed. Thus there is a requirement for high structural strength". As with the minimum weight design driver, the user must decide whether or not to click on the check box corresponding to this rule and introduce the high structural strength design driver into the design process. Moving on now to the low torque and thin section design drivers. The meta rule presented in the corresponding dialog message box for the low torque design driver states "To achieve agility there is a requirement for control surfaces to be capable of high pass rates. This implies the requirement for low torque". The meta rule residing in the message dialog box for the thin section design driver states "To alleviate the effects of shock stall and postpone the drag rise to high Mach numbers there is a requirement for thin sections". In a similar manner to the minimum weight and high structural strength design drivers the user must decide whether or not to click on the appropriate check boxes and introduce the corresponding design drivers into the design process.

It is considered that in the context of designing a foreplane the user will in all likelihood wish to introduce all the above mention design drivers into the design process. Thus, it is assumed in this instance that the user does indeed click on all the design drivers residing in the Agility dialog box. The user, having clicked on all the check boxes in the Agility dialog box should now click on the OK push button to close down this dialog box. The user should now move on to the Supersonic and Subsonic menu options in the Design Driver Selection menu. Selecting these menu options will present the Supersonic and Subsonic dialog boxes. In an identical manner to the Agility dialog box, these dialog boxes present meta rules in message dialog boxes corresponding to design drivers. In this instance, the Supersonic and Subsonic dialog boxes present one meta rule each, both corresponding to the thin section design driver. The meta rules presented by these dialog boxes are discussed in detail in section 9.2.2.2. Again in the context of designing a foreplane it is deemed appropriate to click on the check boxes residing in these dialog boxes. It should be noted that the thin section design driver has already been introduced into the design process when the check box corresponding to thin section was clicked on in the Agility dialog box. As such, the re-entry of the thin section design driver will not have any additional impact on the design process. However, if the design driver corresponding to thin section in the Agility dialog box had not been selected the meta rules residing in the Supersonic and Subsonic dialog boxes could possibly draw the users attention to this potential omission.

Having closed down the Supersonic and Subsonic dialog boxes by clicking on their respective OK push buttons, the user should now select the Core-Competences menu option from the Design Driver Selection menu. Selecting the Core-competences menu presents the user with the Research and Development Input dialog box, see Figure 9.13. As discussed in section 9.2.2.2, the user is required to select one of three possible levels of R&D i.e., by clicking on the appropriate radio button. Clearly, depending on the design problem under consideration anyone of the three levels of R&D could potentially be selected by the user. However, in the context of illustrating typical results output by the ICES prototype, it is considered appropriate that the second level of R&D be selected from the Research and Development Input dialog box i.e., where the development of a new design will be aiming to expand existing competences.

With the second level of R&D selected the user should select the Cost menu option from the Design Driver Selection menu. Selection of the Cost menu causes the Cost dialog box to be displayed, see Figure 9.14. As indicated in section 9.2.2.2, down the left-hand side of this dialog box there are seven check boxes corresponding to seven design drivers i.e., start-up, material, labour, processing time, part reduction, inspection and assembly. Adjacent to these check boxes are seven note pad push buttons. As with the Agility, Supersonic and Subsonic dialog boxes mentioned above these note pad push button provide access to dialog message boxes. These message boxes indicate the meta rules that will be fired if the associated check boxes are clicked on and the corresponding design drivers that will be entered into design process. Again as indicated in section 9.2.2.2, on the right-hand side of the Cost dialog box there are three radio buttons which correspond to the preferred method of manufacture i.e., one-off, batch and mass production. It should be noted that apart from introducing a method of manufacture into the design process the user is also introducing a manufacturing design driver into the design process. This is achieved in an identical manner to the other design drivers discussed in this section.

It is not proposed to work systematically through the meta rules and associated design drivers as was done above in the context of the Agility dialog box as it is considered that the reader should have a reasonable appreciation of the procedure at this stage. However, in the context of illustrating typical results output by the ICES prototype it is necessary to introduce a selection of design drivers in the Cost dialog box into the design process. The design drivers selected to be introduced into the design process in this instance are start-up, material, processing time, part reduction and assembly. These design drivers were selected as it is considered these are those most likely to impact on the design process. In addition, the method of manufacture selected was batch production. This is because most aircraft structures are produced using this method of manufacture. It is important to note that batch production is also a design driver in the context of the design process.

When the user has finished making the appropriate selections in the Cost dialog box the OK push button should be selected to close the dialog box down. The user should now select the Questions menu option from the KBS menu. The selection of the Questions menu causes the Questions dialog box to be presented to the user. The operation of Questions dialog box was discussed in section 9.2.1.3.2. This dialog box asks the user to respond to seven questions relating to the aircraft structure entered in the Structure Identification dialog box. The questions presented in the Questions dialog box are as follows.

Q1. Can the structure or parts of the structure be represented as a single homogenous structure? (y/n)
Q2. Does the structure have an enclosed load bearing internal section? (y/n)
Q3. Will the structure have to support loads such as torsion and bending that lie outside the structural load path of the spars and thus require ribs or comparable stiffening medium? (y/n)
Q4. What is the maximum section wall thickness, in mm?
Q5. What is the minimum section wall thickness, in mm?
Q6. What is the depth of the section, in mm?

Q7. Is it a prime requirement of the structure to be able to dissipate heat? (y/n)

A typical response that a user might conceivably make to the above questions is illustrated in Figure 9.28, this is clarified below.

Q1. Can the structure or parts of the structure be represented as a single homogenous structure? (y/n)

A1. y

Q2. Does the structure have an enclosed load bearing internal section? (y/n)

A2. y

Q3. Will the structure have to support loads such as torsion and bending that lie outside the structural load path of the spars and thus require ribs or comparable stiffening medium? (y/n)

A3. y

Q4. What is the maximum section wall thickness, in mm?

A4. 6 mm

Q5. What is the minimum section wall thickness, in mm?

A5. 3 mm

Q6. What is the depth of the section, in mm?

A6. 150 mm

Q7. Is it a prime requirement of the structure to be able to dissipate heat? (y/n)

A7. n

Having responded to the questions in the Questions dialog box, the user should close down this dialog box by clicking on the OK push button. The user should now select the Best Structure menu option from the KBS menu. Selecting this menu option will present the Best Structure dialog box to the user, see Figure 9.29. The operation of the Best Structure dialog box was explained in section 9.2.2.4. With the Best Structure dialog box displayed the user should click on the 'Best Structure?' push button. This causes the rules residing in the manufacturing and structures rule bases to be fired. It is important for the reader to appreciate that the user inputs made as described previously in this section will dictate the outcome of the rules fired in the manufacturing and structures rule bases. As a result of the firing of these rules the structural types embraced by the ICES KBS are ranked in descending order of preference. The ranked structural preferences of the manufacturing and structures rule bases are displayed in the appropriate list boxes titled Manufacturing and Structures.

In the context of the user inputs indicated in this section, the preferred structure for the manufacturing rule base is spaced x-core. Whilst, the preferred structure for the structures rule base is cellular core. Figure 9.29 shows these structural types display at the top of their corresponding list boxes. As the preferred structural types of the manufacturing and structures rule bases are not in agreement the user must now choose a preferred perspective. That is, view the problem either from the structures perspective or the manufacturing perspective and then fire the second rules

accordingly. This aspect of the ICES prototype was discussed in section 9.2.2.4 and the underlying methodology supporting this concept is outlined in section 7.2.4.2 of Chapter 7. There is no practical reason why the user should not select either the manufacturing or the structures perspective in this instance. However, as BAe MA&A have more experience with spaced x-core than cellular core and the level of R&D selected was aimed at expanding existing competences it would be appropriate for the user to select the manufacturing perspective i.e., spaced x-core.

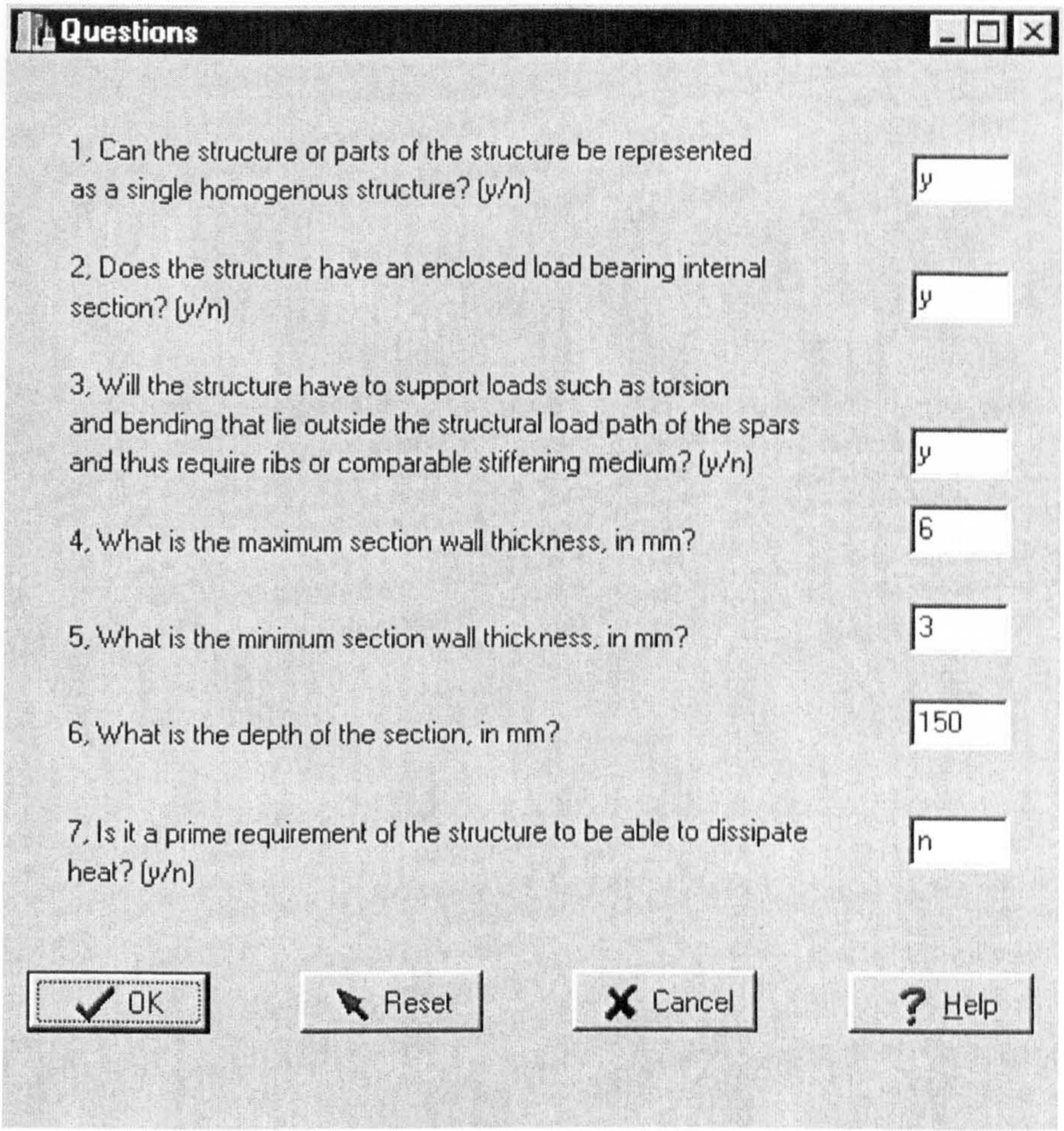



Figure 9.28, The Questions Dialog Box Displaying User Responses

Having decided to take the manufacturing perspective, the user should now select the Spaced X-core push button listed under the heading Structures in the Best Structure dialog box. Clicking on the Spaced X-core push button will present the Spaced X-core Structures Secondary Rules dialog box to the user, see Figure 9.30. This dialog box presents the secondary rules that the user must fire in order to make standard x-core the preferred structure for the structures rule base. It can be seen from Figure 9.30 that only one secondary rule is applicable in this instance. The rule states "Is it possible for the increase in structural mass necessary to carry out-of-plane loads using an x-core structure to be allowed for within the design?" The user must now click on the check box adjacent to the rule text to facilitate the firing of the rule. Having fired the secondary rule the user should close down the Spaced X-core Structures Secondary Rules dialog box by clicking on the OK push button. The application will now return the user to the Best Structure dialog box. The user should now click on the 'Best Structure?' push button again. The spaced x-core structure

which was previously listed second in the Structures list box will now be seen to move to the top of the list. As the preferred structures of the manufacturing and structures rule bases are now in agreement, the spaced x-core structure will now be declared as the best structure in the best structure edit box immediately below the 'Best Structure ?' push button.

With the spaced x-core structure declared in the best structure edit box, the user should now click on the note pad push button immediately below the best structure

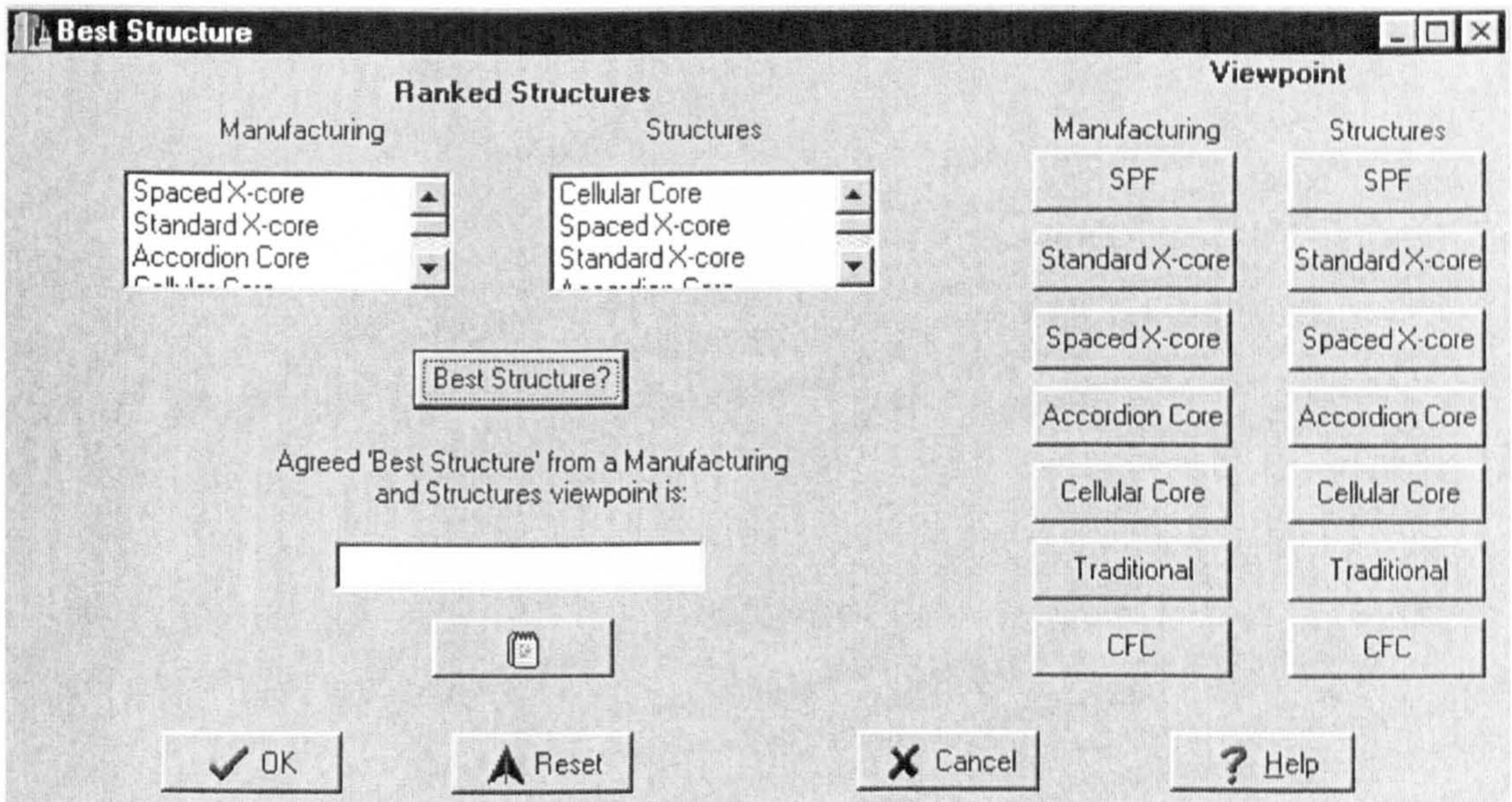



Figure 9.29, The Best Structure Dialog Box Showing the Manufacturing and Structures Rule Bases Preferred Structures

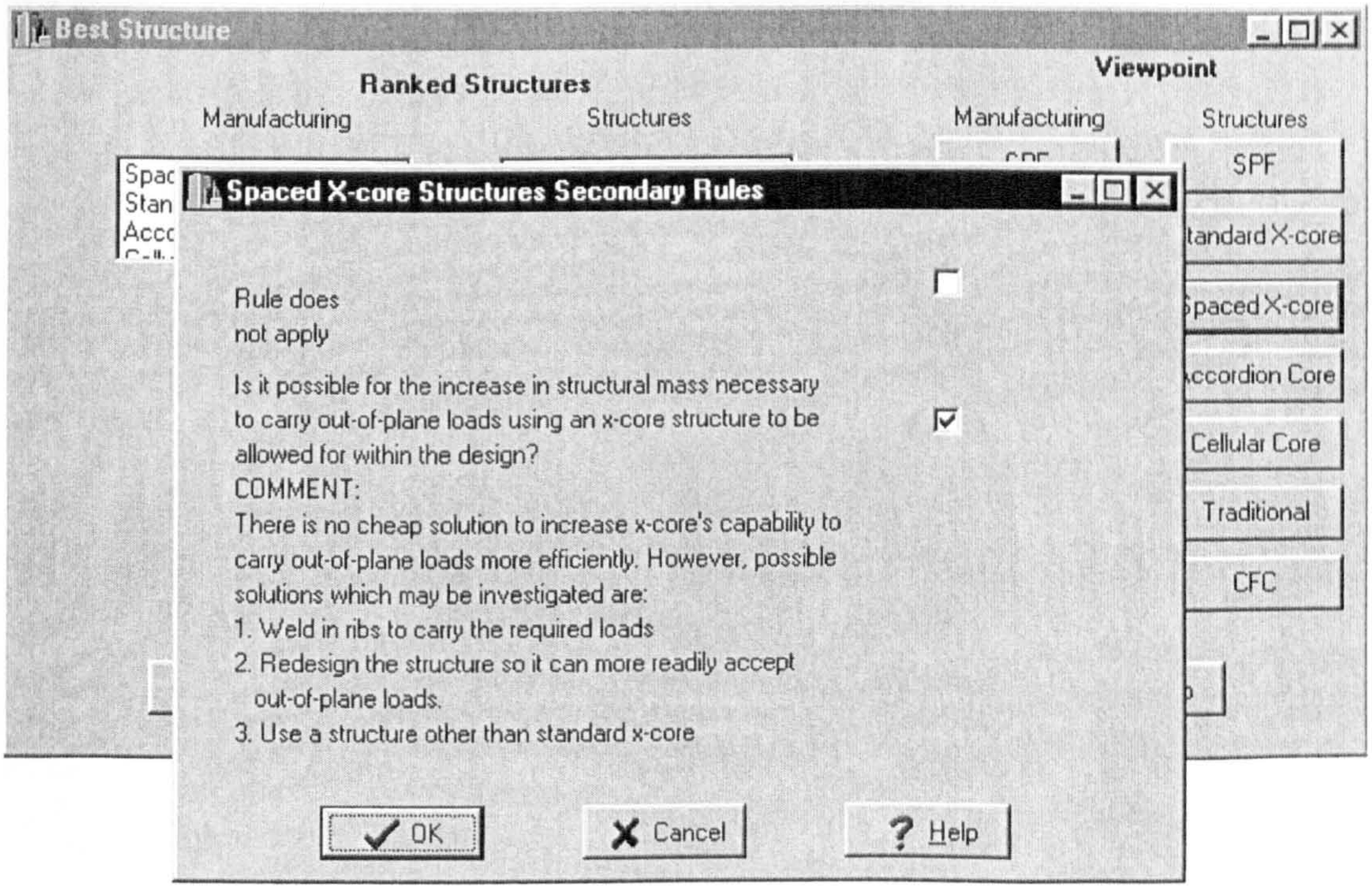



Figure 9.30, The Spaced X-core Structures Secondary Rules Dialog Box

edit box. The Best Structure - Audit Trail dialog box will now be presented to the user, see Figure 9.31. The operation of the Best Structure - Audit Trail dialog box is discussed in section 9.2.2.4. The user should click on the Spaced X-core Audit push button in the Best Structure - Audit Trail dialog box, this being the only push button that is enabled, again see Figure 9.31. When the Spaced X-core Audit push button is selected the Spaced X-core Audit Trail dialog box is presented to the user as illustrated in Figure 9.32. The text in this dialog box explains to the user why spaced x-core is considered to be the best structure; this being in the context of the system responses made by the user leading up to defining the best structure.

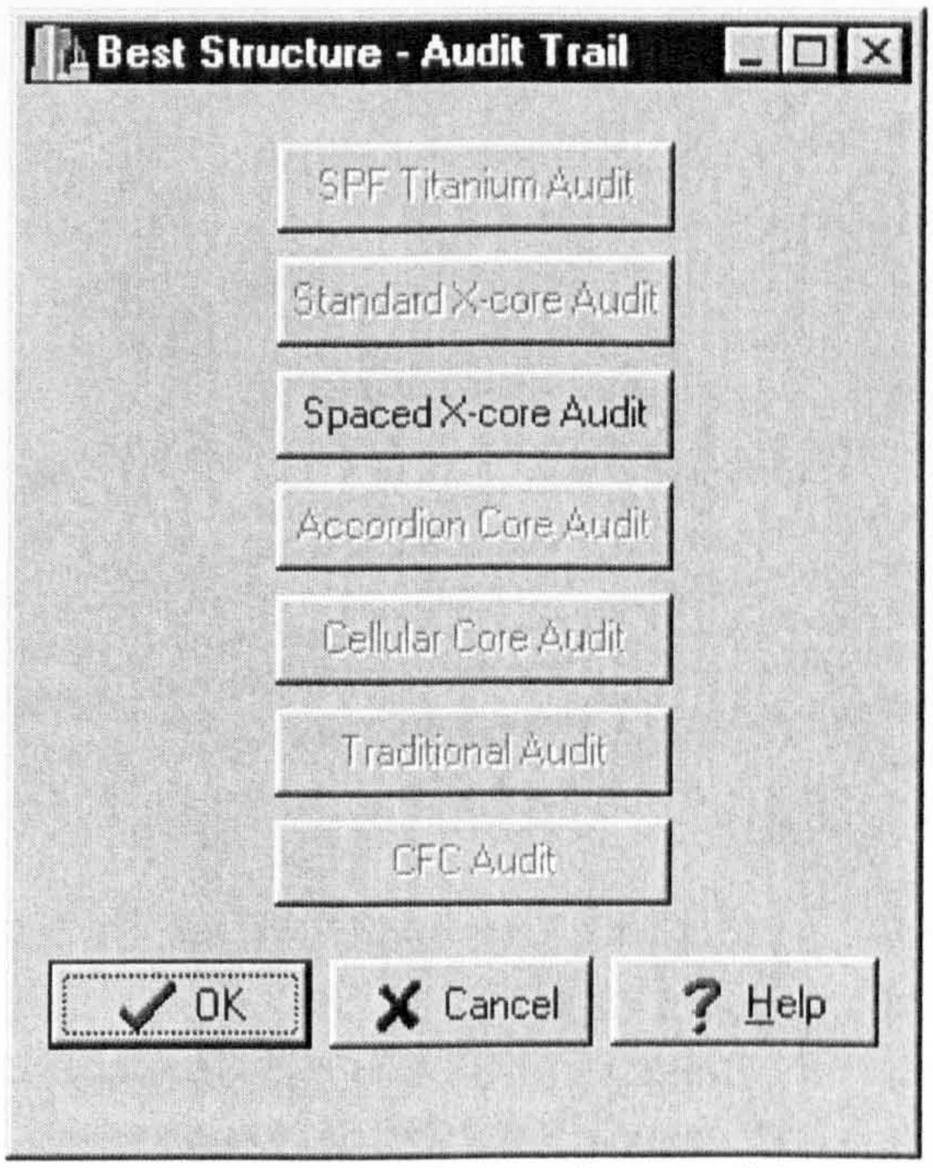



Figure 9.31, The Best Structure - Audit Trail Dialog with the Spaced X-core Push Button Enabled

This section has taken the reader in a logical step-by-step manner through an example run of the ICES KBS. In this instance, spaced x-core was presented as the best structure. The following section follows on from this one by providing an example run of the ICES case base. In the example run presented in the following section it is assumed that spaced x-core has been identified by the KBS as the best structure.

### 9.3.2. The ICES Case Base Results

As indicated in section 9.2.3.2, the ICES case base can be queried into two ways. Firstly, the user can identify those cases in the case base which possess the best structure as identified by the ICES KBS. Secondly, through the entry of known specifications the user can identify the best case residing in the case base whose specifications correspond most closely with those entered. This second mode of querying the case base is supported by the jury technique as discussed in section 7.3.3 of Chapter 7 and section 9.2.3.2.2 of this chapter. It is proposed to start the discussion in this section by first presenting the results output by the case base in the context of querying the case base with respect to the best structure as defined by the KBS component of the ICES prototype.

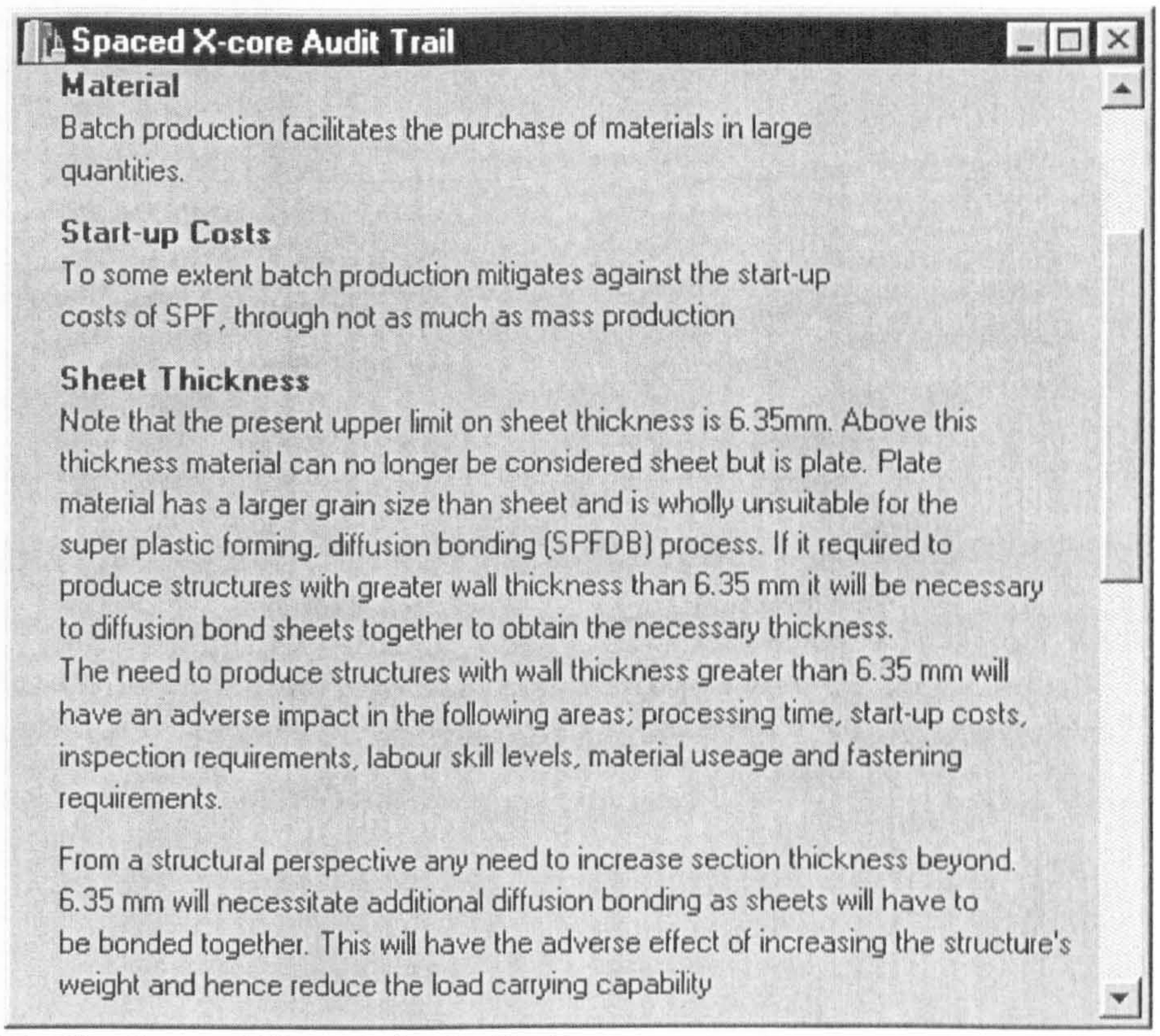



Figure 9.32, The Spaced X-core Audit Trail

It will be recalled from the previous section that spaced x-core was identified by the KBS as being the 'best structure'. In order to identify the case(s) in the case base which utilise this structure the user should interrogate the case base in the following manner. The user should first select the Case Query and Jury menu option from the Case Base menu. This will present the Case Query and Jury window to the user. From the Case Query and Jury window the user should select the Best Case menu option from the KBS menu. This is all the user input that is required. The case base now initiates a search of all the foreplane cases; looking for foreplanes that are constructed using spaced x-core. When a suitable case is found it is displayed to the user. Figure 9.33 illustrates the output presented by the case base after this search. The foreplane case presented is taken from the European Fighter Aircraft (EFA). The EFA foreplanes are manufactured using spaced x-core.

This discussion will now focus on the second method of querying the case base as indicated above i.e., entering specifications and applying the jury technique. In the manner indicated above the user should cause the Case Query and Jury window to be displayed. With the Case Query and Jury window displayed the user should enter the known specifications in the column of edit boxes on the left-hand side of the window, see Figure 9.34. Clearly, the user may input any specifications he or she likes. However, in the context of this illustration the specifications entered are as follows.

**Case Base Specifications Input**

| | |
|---|---|
| Component | foreplane |
| Structure Type | |

| | |
|---|---|
| Max. Pos. Shear Load kN | 330 |
| Max. Neg. Shear Load kN | -161 |
| Max. Pos. Bend. Load kNm | 340 |
| Max. Neg. Bend. Load kNm | -162 |
| Max. Pos. Torque Load kNm | 96 |
| Max. Neg. Torque Load kNm | -60 |
| Tensile Yield Stress MPa | 1000 |
| UTS MPa | 900 |
| Young's Modulus MPa | 110700 |
| Density $Mg/m^3$ | 3.2 |

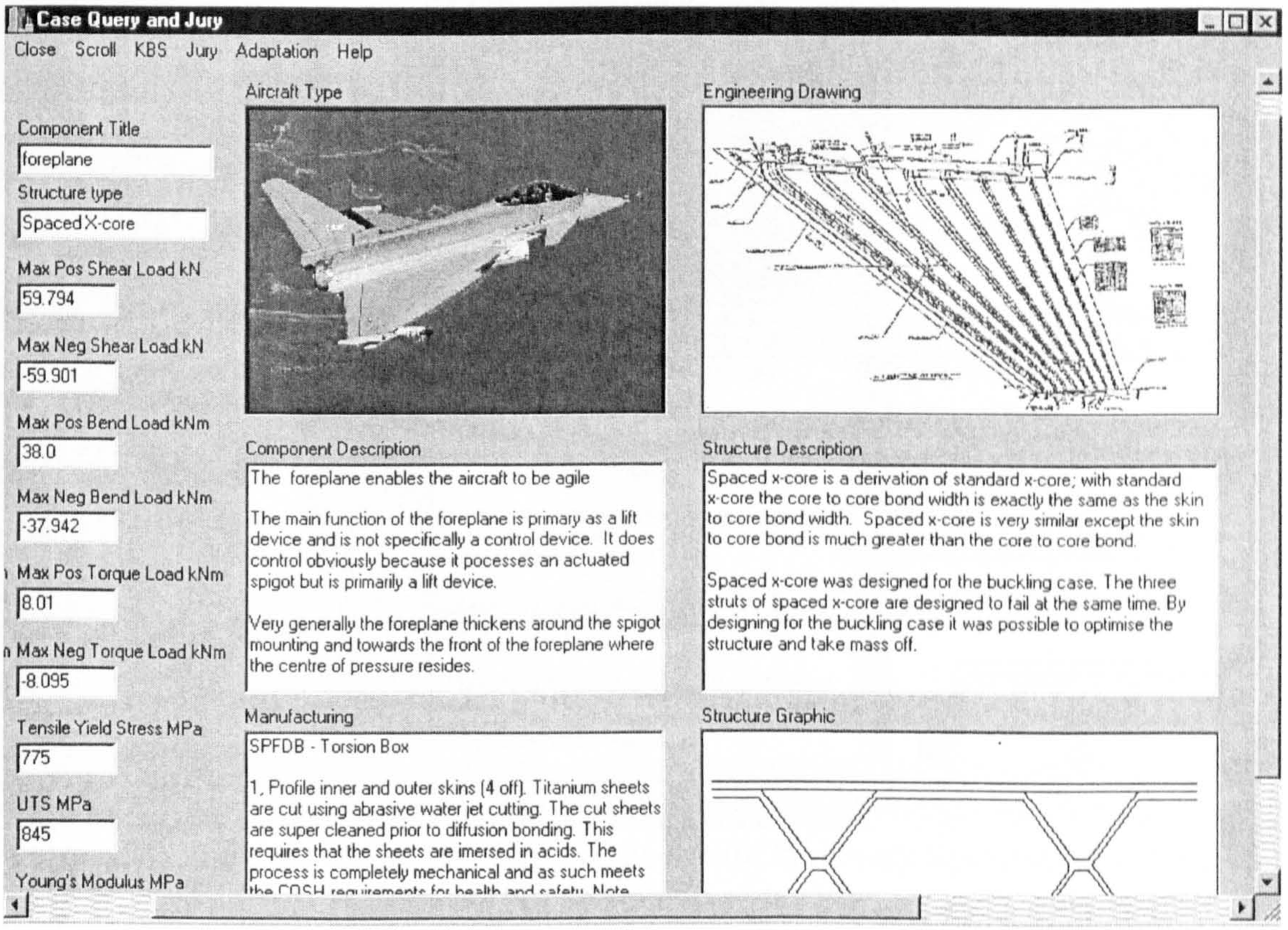



Figure 9.33, The Case Query and Jury Window Displaying the European Fighter Aircraft (EFA) Foreplane Design Case

Having entered the above specifications, the user should select the Best Case menu option from the Jury menu. As indicated in section 9.2.3.2.2, the Best Case menu option causes the best case to be derived through nearest neighbour matching. In the context of the above entered specifications, the case presented as the best case is the Experimental Aircraft Programe (EAP) foreplane case, see Figure 9.35. With this best case presented the jury technique should now be applied. The user should now study all the information presented by the EAP case and in particular the Defence field. Having studied the case, the user should now select the Next Best menu option from the Jury menu. This menu option again applies the nearest neighbour matching technique to find the case which is the next best case with respect to the input specifications. The next best case presented in the Case Query and Jury window in this instance is the EFA foreplane case, see Figure 9.36. Again, as with the EAP foreplane case the user should examine this case, paying particular attention to the

Defence field. The user may cycle back and forth between the best and next best case as often as desired. Clearly, there are good arguments that may be put forward for both the EAP and EFA foreplane cases. The Defence field for the EAP foreplane case highlights the advantages of composite materials as being; their high stiffness to weight ratio; that they are normally linear to failure and as such no plastic flow occurs; that they lend themselves to a wide range of manufacturing methods unlike some alternative structures; BAe MA&A has considerable experience with the manufacture of a range

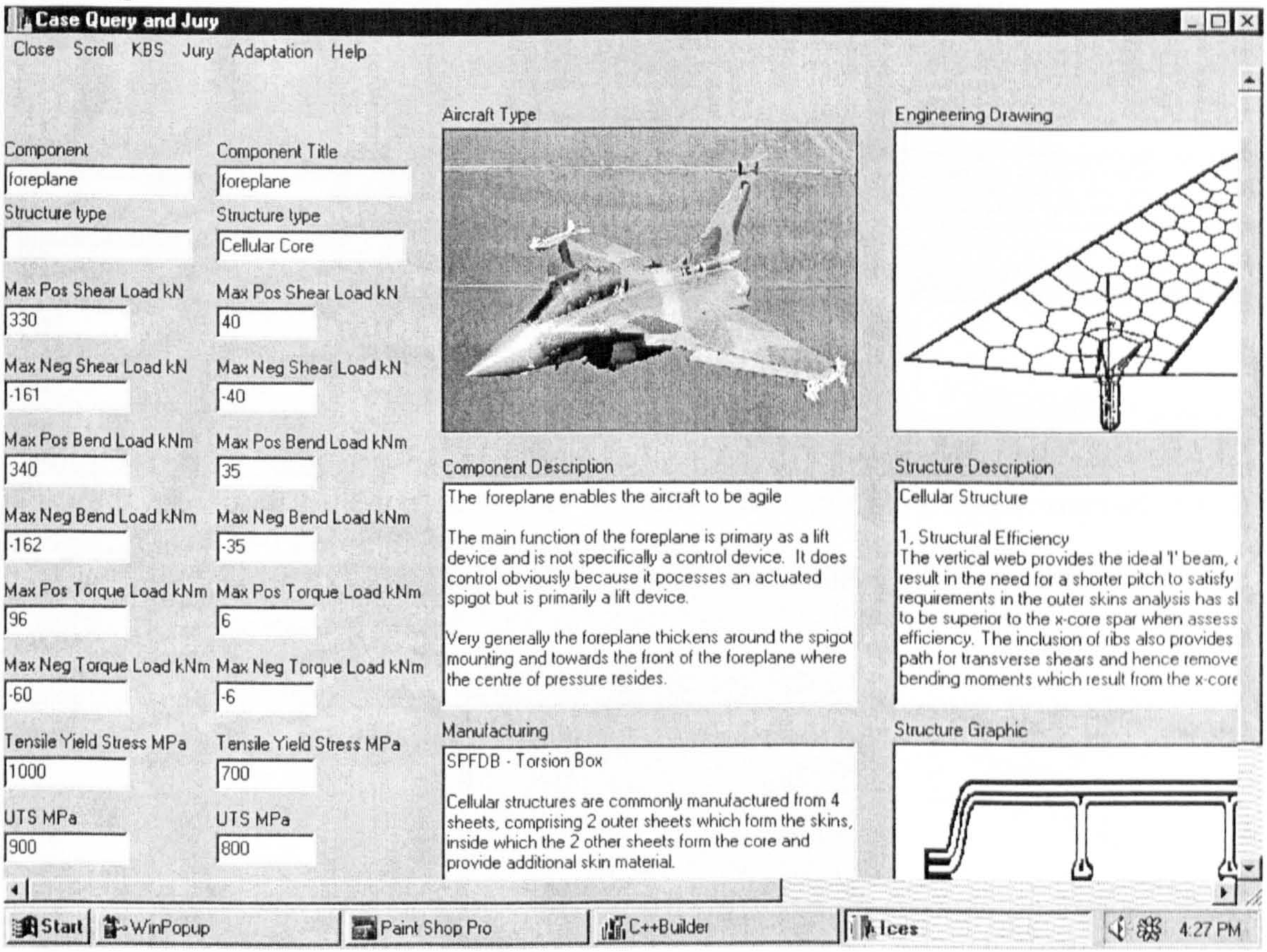


Figure 9.34, The Case Query and Jury Window with the User's Specifications Entered

of composite structures. The Defence field for the EFA foreplane case highlights the advantage of spaced x-core structures as being; that their is now a proven track record with this advanced structure; SPF/DB structures are superior to composite structures for transmitting point loads; the structure has been proven to be extremely strong in destructive tests.

If the user prefers the EAP foreplane case then no further action is required. However, if the user prefers the EFA foreplane case the user should cycle back to the EAP foreplane case and select the Reject Case menu option from the Jury menu. The selection of the Reject Case menu option will cause the EAP foreplane case to be rejected. The user should now select the Best Case menu option; the EFA foreplane case will now be presented as the best case. It should be noted that there will be now a new next best case which should be compared with this case. The process of comparing the best case with the next best case will continue as described until the user decides that the best case presented is indeed the best case. If the user feels that no case presented is suitable then he or she may consider applying case adaptation as described in section 9.2.3.2.3.

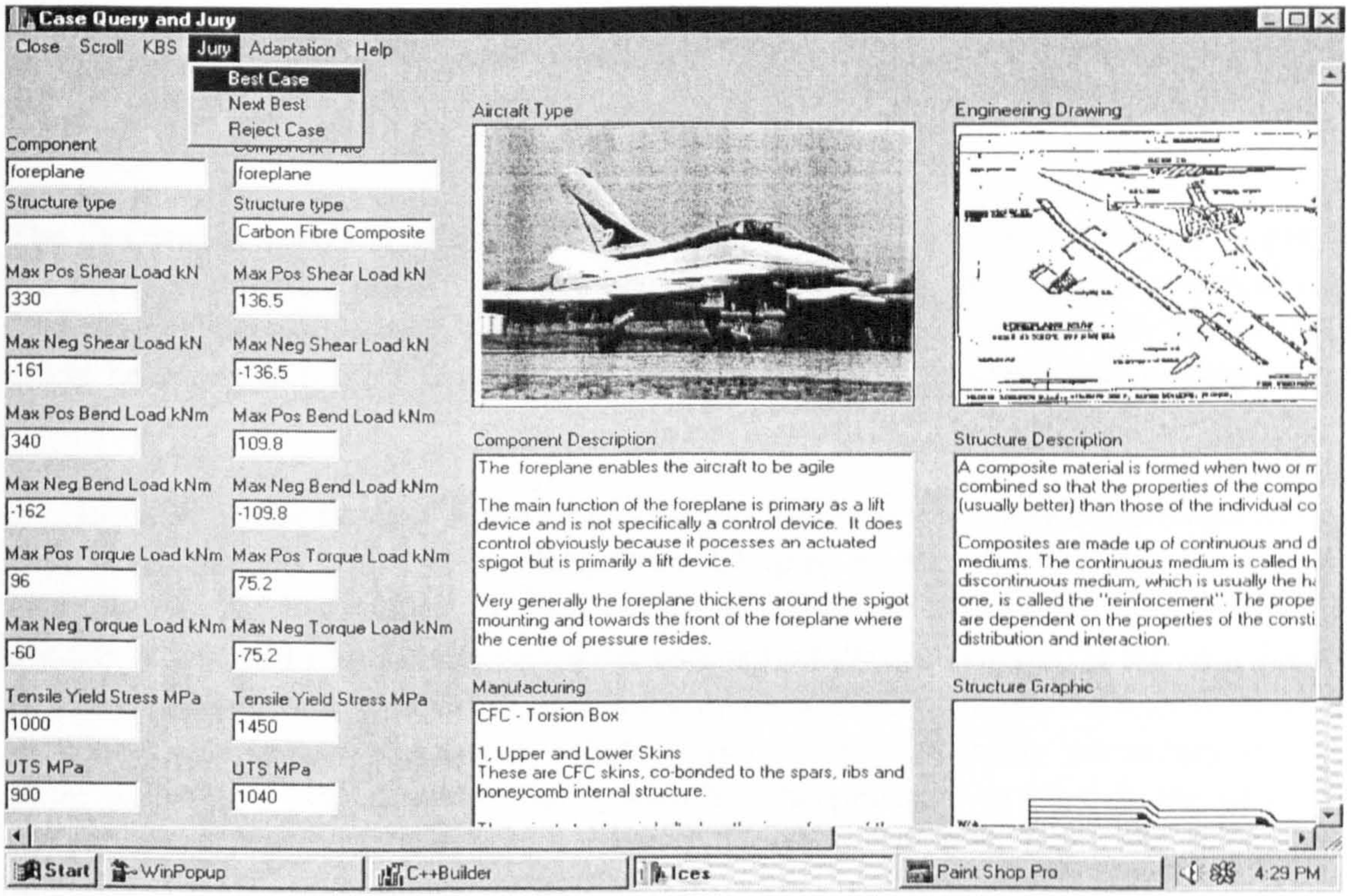



Figure 9.35, The Case Query and Jury Window Displaying the Experimental Aircraft Programme (EAP) Foreplane Case as the Best Case

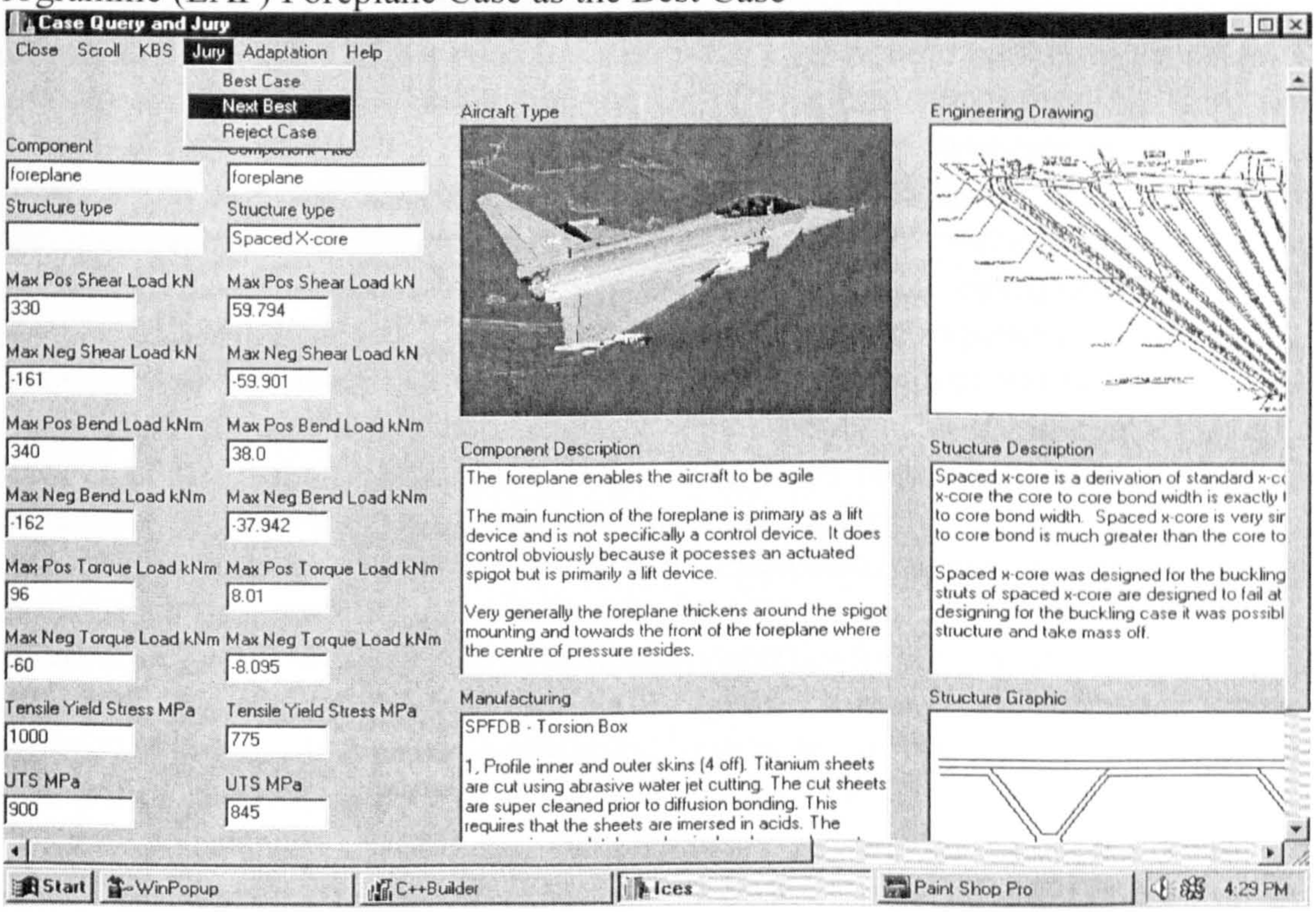



Figure 9.36, The Case Query and Jury Window Displaying the EFA Foreplane Case as the Next Best Case

# Chapter 10

## Results Discussion

The discussion in this chapter starts by looking at the quality of the results output by the ICES prototype. That is, are the results valid and are they of any practical use? The chapter then leads on to discuss areas of the prototype as it currently exists which could be improved. Here focus is placed on the manner in which the prototype was coded.

Looking at the quality of the results output by the ICES knowledge-based system (KBS) component. As can be seen from Chapter 9, the ICES KBS successfully identifies the 'best structure'. In the process, it ensures that the user embraces all the aspects that need to be considered in the conceptual design process. If the user fails to consider manufacturing or R&D requirements the system prompts the user to make an input in these areas. In addition, where it is necessary to apply secondary rules in order to derive a best structure, the system has the potential to provide the user with insight into what is required to arrive at the best structure from the perspective selected.

Turning now to the quality of the ICES case base output. It is recognised that the information provided by the design cases in the case base is not comprehensive. The cases do not cover all the facets of the design of foreplane structures. To a large extent they provide the user with a superficial appreciation of the foreplane in question. This is not to belittle the information that is provided. The information in the cases provides the user with an appreciation of the all the pertinent design aspects, particularly in the areas of manufacturing and structural configuration. However, specific detailed design information relating to specific components of the foreplane cases is not available e.g., information relating to the adhesives and fasteners in design cases using composites is not available. In addition, it is appreciated that in the context of the design cases which reflect foreplanes constructed using carbon fibre composite (CFC) materials, the case base is lacking with respect to material specification information. This is because the composite design cases in the case base do not take into account the orthotropic nature of composite materials. That is, an orthotropic body has usually three different properties in three mutually perpendicular directions at a point in the body and has only three mutually perpendicular planes of material symmetry at a point in the body. That is to say the material properties are a function of the orientation at a point in the body. The information presented in the CFC design cases correspond only with loads applied parallel with the fibre direction i.e., the principal axes. Clearly, there is considerably more information that could be presented e.g. loads acting perpendicular to the fibre direction. There are three reasons why the information relating to composite foreplane design cases was lacking in this area. Firstly, this information was not always readily available. Secondly, due to the time constraints imposed on this study it was necessary to prioritise the information that cases were going to display. That is, it was considered desirable that all the design cases in the case base should display relatively equal amounts of information. Thus, enabling the user to make a fair assessment of one case opposed to another. Clearly,

if one case were to present vastly more information than another the case base would be rather bias. Finally, the Paradox relational database used in this study was not the ideal way of representing design case knowledge. An object oriented database would have been much the preferred method of representing CFC design case information.

Having acknowledged that the information contained in the design cases in the case base is not comprehensive it is necessary to appreciate that to fulfil the objectives of the case base the information need not be wholly complete. That is, the purpose of the case base is to present sufficient information to enable the user to make a reasoned choice with respect to which design case most closely fulfils his or her needs. In this respect the ICES case base functions correctly. Clearly, when the user selects a particular case more detailed information can be obtained from other sources e.g., engineering drawings residing on the CATIA computer aided design (CAD) package for instance.

As indicated in section 8.3.5 of Chapter 8, it was decided to use a relational database i.e., Paradox, instead of an object oriented database. Whilst, as stated in section 8.3.5 it would have been preferable to use an object oriented data base, the practicalities of implementing such a database in conjunction with C++ Builder where prohibitive. Specifically, there was not a visual database software package which could be interfaced with C++ Builder. Therefore, however undesirable, it was apparent that a relational database would have to be utilised. Unlike the object oriented database which possesses a natural hierarchical structure which can be made to map closely to the foreplane design structure or any other structure for that matter, the relational database has a flat table structure which bears no natural relationship at all to any real design structure. A relational database does not naturally break down a structure or design case into a prioritised format. For example, a field in a relational database table corresponding to manufacture has no more importance than a field that supports the specifications relating to a type of fastener. It is therefore necessary in the context of the visual environment that C++ Builder supports for the developer to decide which fields to display to the user. It should be noted that it possible to overcome the flat nature of relational databases by representing design case information in a hierarchy of dialog boxes. Clearly, it will be necessary to carefully plan any such hierarchy so that the case base retains its generic nature and the user can appreciate where all the information within a case resides. While such a hierarchy bears some similarity to the frame-based approach of structuring case knowledge as described in section 4.4 of Chapter 4, it should be noted that a hierarchy of dialog boxes representing case information does not support inheritance like the frame-based approach does. The frame-based approach of supporting case knowledge can only be achieved in an object oriented environment.

In the context of the case base presented in this thesis it could be argued that the information could have been better presented by employing a hierarchy of dialog boxes. However, it was considered that all the information was adequately displayed in a large scrollable window such as the New Case or the Case Query and Jury window. Clearly where the cases contained considerably more information a single scrolling window would not be practical and a hierarchy of dialog boxes would have to be employed.

Even allowing for the potential of representing aircraft design cases with a hierarchy of dialog boxes and the scrolling capability of windows there is a practical limit to how many fields can be sensibly displayed to the user with a standard computer monitor. Clearly, the user does not want to have to perform an inordinate amount of scrolling in order to see case information. In addition, if the user has to perform excessive scrolling to see the particular item of information that he or she is interested in then it is likely that the user will quickly become disenchanted with the software. There is also the problem that because a large quantity of information is out of view, the user will not know what information is available and where in the scrolling window it resides; also there is the potential for the user to lose orientation i.e., become lost in the application. As indicated above, the problem of presenting case information in a readily accessible format can possibly be overcome by presenting design case information using a hierarchy of dialog boxes and windows. However, if the hierarchy of dialog boxes and windows is too extensive the user may again experience difficulties. For example, there may be problems where the user wishes to cross reference with information placed in other dialog boxes and windows in the hierarchy.

There appears to be scope for a formalism to be documented of how a large quantity of design information should be best displayed within the limitations of a standard PC monitor using visual tools. As discovered when coding the ICES prototype this aspect is very much left to the developers discretion. Indeed this argument can also be extended to all computer aided engineering (CAE) software. That is, as discussed in Chapter 8, it was initially intended to place ICES within the CATIA CAD package; there are no guidelines as to where this tool should have been placed within CATIA. The only limitations on where ICES could be placed were dependent on the underlying structure of CATIA. The point that the Author is making is better illustrated if one was to consider placing ICES in another proprietary CAD package. It is very unlikely that the user, familiar with the location of ICES within CATIA, could intuitively locate ICES within this other CAD package. Ideally, it would be desirable to be able to come to all forms of PC and workstation based CAE software which handles a large range of structural information and know within reason how information will be presented even before the software is accessed. Taking, industry standard office software as an analogy, once the user has a 'feel' for one software package he or she knows intuitively where to look for the same information on comparable packages made by a different proprietor. Clearly, it is appreciated that CAE software is vastly more complex than that used in the office environment. However, there is no practical reason why steps cannot be taken to generate this common feel.

The advantages of documenting a formalism of how design information should be displayed in a PC and workstation environment would be considerable. The time it takes for a new user to become familiar with a design software package would greatly be reduced. When proprietary design software is upgraded, the upgraded software would be guaranteed to have the same feel as before. From the Author's perspective as a software developer, there would be an intuitive appreciation of how a software package such as ICES should look and feel. At the present time the only guide that exists is the basic Windows format i.e., File Edit etc. Clearly, Windows was designed

with the office environment in mind and not the design or engineering environment. It is considered that this fact underlines the point being made in this discussion; should engineering software developed by an individual really have the feel of office software e.g., a text editor, for the sole reason that there is no recognised industry standard for engineering software available? Currently, if the ICES methodology as presented in this thesis were to be coded by any number of different people there could be conceivably be any number of solutions each with its own individual feel. Clearly, it would be desirable, with in bounds, that each version of ICES have the same feel.

It could be argued that such formalism for displaying design and engineering software could stifle creativity. It is the Author's opinion that this argument is flawed. This is because the underlying creative ideas that support software development should come about and be evolved independently of the coding process. The process of coding is just the transferring of the developer's ideas into a computer language. Taking this thesis as an example; the concepts presented in Chapter 7 where conceived independently from the coding process. However, it is acknowledged that while the concepts where being developed by the Author there was a constant awareness of the requirement to be able to place the concepts in a software environment.

Due to C++ Builder not supporting true object oriented programming, the coding of the KBS rules was a more time consuming process than it might otherwise have been. If the prototype were to be expanded then the transferring of ICES to another software platform should be considered e.g., Microsoft Visual C++. In terms of the structure of the code it is considered that the code could be improved if the KBS rules acting on the system were kept separate from the main body of the code. That is, making the software more modular and thus, easier to edit at a later date. The reason for putting the code supporting the KBS rules in the main body of the code was because they are generic and could be applied to all aircraft structures. It was acknowledged that any non-generic rules would have to be kept separate from the main body of the code. However, in retrospect, it is considered that both the generic and non-generic rules should be kept separate from the main body of the code.

# Chapter 11

# Conclusions

This conclusion is divided into three sections. The first section looks at the work embraced by this study and concludes on the principle constituent components as appropriate. The second section looks at the ICES methodology and subsequent prototype as developed in this study and discusses the quality of the work i.e., is the end product any good? The third section discusses how ICES could be further developed. In particular, in the context of developing a 'fully fledged' system and adding to existing capability to support new functionality.

## 11.1. Study Conclusion

This study has presented a new design methodology which has been developed around the requirements of the military aircraft industry. Specifically, the methodology successfully addresses the difficult management problem of capturing, storing and subsequently re-using company product knowledge. The detailed research documented in this thesis has focused on the conceptual design area of this methodology. This detailed research has resulted in the development of a further methodology for the conceptual design arena. This methodology is called the intelligent conceptual engineering system (ICES). The concepts presented within the ICES methodology have been placed in a prototype design decision support tool coded using the C++ Builder rapid application development (RAD) software package.

The ICES methodology presented in this study has applied the knowledge-based system (KBS) and case-based reasoning (CBR) technologies to tackle two 'real' problems. Firstly, the KBS component of the ICES methodology has been shown to be able to identify a 'best structure' from a manufacturing and structures perspective. The KBS incorporates numerically justified design drivers into the decision making process. The KBS component has been shown to be able to draw in the R&D decisions that need to be made in the design process at an early stage. The ICES software draws the designers attention to the level of R&D input that will be required for certain structural types and manufacturing scenarios.

Secondly, the CBR component of the ICES methodology has been shown to be able to identify the 'best' past design case residing in a database of previous design cases. This has enabled relevant past design information to be drawn into the design process. Thus, facilitating it being available to assist with the solution of the current design problem. The ICES methodology has applied case-based reasoning (CBR) in a new way. This study has introduced a new concept to CBR called the jury technique. This technique is able to tackle large cases where knowledge may be incomplete. The technique is not reliant on there being a large number of cases. The concept underpinning the CBR component of the ICES methodology is that it is a requirement of the system that the case presented by the system as being the 'best case' defend itself with respect to the next best case. The user(s) are the jury and decide whether the case presented as being the 'best' is indeed superior to the next best case.

In terms of improving knowledge management this study has applied a three phase process. The first phase as discussed in Chapter 3 took a strategic view of the current design process utilised by BAE SYSTEMS. Here the study identified the types of knowledge that are used within the design process and how it would be desirable for this knowledge to be channelled. The second phase, as discussed in Chapter 5 identified the appropriate technology to facilitate the capture and subsequent reuse of product knowledge i.e., knowledge-based system and case-based reasoning. The third and final phase, determines in detail how the technology selected should operate; this aspect is embraced by ICES as documented in Chapter 7.

It is considered that this three phased approach to knowledge management applied in this study represents a logical method for determining the most appropriate means to capture company product knowledge. The approach, through applying a range of levels of resolution to this difficult management problem, systematically identifies the most appropriate mechanisms to facilitate the processing of company product knowledge.

## 11.2. The Quality of ICES

The customer in the context of this study is BAE SYSTEMS. With this in mind, it was necessary to present the final ICES software prototype at British Aerospace, Warton. As such, the second prototype was presented in front of a range of engineering staff at British Aerospace, Warton in July 1998. Following a presentation which alluded to the salient points of the ICES methodology, the ICES prototype was demonstrated. For the prototype to be considered to have any worth it was necessary that the British Aerospace engineering staff recognise the system to have potential in the real world environment. The demonstration given was based on the example provided in sections 9.4.1 and 9.4.2 of Chapter 9.

Following the demonstration of the ICES prototype the forum was open to discussion. The general consensus among British Aerospace engineering staff present at the demonstration was that the ICES prototype as a technology 'prover' had considerable merit. In the context of being a potential tool for providing assistance in the conceptual design arena, it was considered that the prototype could well assist designers to focus in on potential designs and structural types. An area of application for the ICES prototype which the Author had not considered was the potential for the ICES prototype to be used as a teaching/training aid for new members of staff at British Aerospace. As both the ICES prototype components i.e., KBS and case base, provide explanations for the reasons why certain structures and design cases are preferable, this aspect could be applied in a teaching/training forum. To fulfil this possible role, it is considered that the ICES prototype would have to embrace an increased amount of explanatory text. As it was appreciated that the ICES prototype was only a technology 'prover', interest was expressed with respect to the possibility of developing a full system. The likelihood of a full system based on the ICES prototype being developed is discussed in the following section. In the context of the response to the ICES prototype received from the customer i.e., British Aerospace engineering staff, it is considered that the prototype is a success and satisfactorily fulfils the role for which it was designed.

In conclusion, the research project documented in this thesis represents part of BAE SYSTEMS's research into the use of artificial intelligence (AI)technology for the management of knowledge. In strategic terms, this study has proven the viability to BAE SYSTEMS of employing AI computer based tools for the capturing and delivering of product knowledge to the points of need.

## 11.3. Further Development of the ICES Prototype

### 11.3.1. Developing a Full System on the Basis of the ICES Prototype

As indicated above, interest was expressed during the presentation of the ICES prototype of the possibilities of taking the prototype further and developing a fully fledged system. To be able to implement concepts presented in this study it will be necessary for a corporate strategy for knowledge management to be in place i.e., as outlined in Chapter 1. The implementation of the ICES methodology on a larger scale would be much easier to achieve where knowledge is managed as a separate discipline within the organisation. That is, knowledge management is not solely embedded within existing separate functional disciplines. With such a scenario, projects such as the one presented here have a natural course of development beyond the functional department where they were initially instigated. With a corporate knowledge strategy in place everybody who needs to know about a project is naturally informed of its existence and able to make input. In addition, where knowledge is managed on a company wide level, the benefits of studies such as this one will be maximised. This is because it is conceivable that people will see an application for the concepts presented in the study beyond those envisaged by the Author.

### 11.3.2. The ICES Prototype - Further Work

As indicated throughout this study, the retention of and subsequent re-use of product knowledge and the management of intellectual capital is an issue that confronts the entire military aircraft industry and not just BAE SYSTEMS. The pressure to reduce costs and yet maintain innovative design is the rational behind the need to manage knowledge effectively. The underlying reasoning behind this study was that if a methodology could be developed that facilitated the better management and processing of company product knowledge then this would enhance BAE SYSTEMS's sustainable competitive advantage. It was acknowledged that traditional design methods fail to foster knowledge retention; this study through the application of artificial intelligence technology has presented a new design methodology whose underlying aim is to address this deficiency in traditional design methods. That is, to facilitate the capture of product knowledge and deliver it to the points of need in the design process. The ICES prototype developed during this study has successfully demonstrated the potential for capturing product knowledge and outputting this knowledge in a useful format. It is important to appreciate that the work covered by this study embraces a proof of concept research programme and was never intended to be totally comprehensive in all areas. However, during the evolutionary development of the methodology and subsequently the prototype it became apparent that there were areas of the study which lend themselves to further development. The areas where the

Author would like to see further development of the methodology and prototype are discussed below:

- The ICES methodology presented in Chapter 7 focused on drawing knowledge in from the manufacturing and structures disciplines. Clearly, there is potential to expand the methodology such that it embraces knowledge from other areas. For example, aerodynamics, aeroelastics, and non-structural mass knowledge. These disciplines where alluded to in Chapters 3 and 5. However, in the context of the time scale of this research project there was insufficient time available to permit these disciplines to be encapsulated within the ICES methodology. If the prototype were to be further developed it would be desirable that these important areas be drawn into the system. In addition to focusing on other disciplines the ICES methodology should also focus attention and attempt to accommodate financial information. This is an important aspect which the methodology currently does not embrace.

- An obvious area where it would be desirable to expand the ICES prototype is in the context of enabling other structures to be driven through the prototype. Whilst, the prototype as it exists is generic, it needs to be developed further in order for it to truly support a full range of aircraft structures. At the present time there exists just sufficient capability to enable a foreplane structure to be driven through the system. Specifically, future development should add new rules to the system and a much expanded questioning process. There is certainly potential for the system to draw much more information from the user before any attempt is made to identify a best structure or case.

- Another area where the work presented in this study could be expanded would be to research the possibilities of applying the methodology presented within other industries e.g. ship building and motor vehicle manufacturing.

- Turning to the ICES case base. As indicated in section 7.3.3 of Chapter 7, the case base presents the potential risks associated with selecting any particular design case from the case base. It is considered that future work should include a full risk assessment process. Future work should research the available means of attributing some form of numerical value to the level of risk and explore the use of different weighting formats with a view to producing more accurate results. If the case base facilitated the adding of new information to design cases the levels of risk and the associated numerical value assigned to the risk could possibly then be updated accordingly.

- The Design Driver Ranking Technique (DDRT) documented in section 7.2.2 of Chapter 7, has extensively used numerical values for weighting. Future work should explore the use of different weighting formats with a view to producing more accurate results.

- The detailed research encapsulated within the ICES methodology and implemented within the software prototype only embraces the conceptual design area of the new design methodology presented for BAE SYSTEMS in Chapter 5 of this thesis. As

such, it would be desirable to implement the artificial intelligence (AI) capability with respect to the finite element analysis (FEA) and structural optimisation areas of the methodology. In terms of feeding the output from this analysis area into the case base, it is considered that there is possibly a role for genetic algorithms here. It is considered that the implementation of AI technology in the area of FEA and structural optimisation would constitute a further significant research project in its own right.

# References


[1]
British Aerospace Chooses Catia for Design of Advanced Military Aircraft with a 13 Million Pound Order.
http://www.uk.ibm.com/releases/april96/96092.htm

[2]
Augustines's Laws
by N.R.Augustine
Published by the American Institute of Aeronautics, Inc
1983

[3]
A Strategy for Manufacturing the Next Generation Strike Fighter
by E.Wyatt.
JS Program Office
10th January 1997

[4]
Joint Strike Fighter Manufacturing Demonstration (JMD)
by A.E.Herner and J.K.Rowland
Hughes Aircraft Company
October 1997

[5,6,7,12]
The Knowledge-Creating Company: The Learning Imperative, Managing People for Continuous Innovation.
by I.Nonaka
Harvard College
1990

[8]
Coming of Post-Industrial Society: A Venture in Social Forecasting.
by D.Bell
Basic Books
1976

[9]
The Second Industrial Divide: Possibilities for Prosperity
by M.S.Piore and C.F.Sabel
Published by Basic Books
1984

[10]
Intelligent Systems
by J.Knightly
Mortgage Banking
May 1996

[11]
Intellectual Capital, How to Build it, Enhance it, Use it
by W.J.Hudson
Published by John Wiley & Sons.
1993

[13]
The Knowledge Advantage
by L.Prusak
Strategy and Leadership
March/April 1996

[14]
Will the Real CKO Please Stand Up?
by W.W.Leake
IIE Solutions
1996.

[15,17]
Strategic-Level Knowledge Management.
by I.Filby.
AIAI, The University of Edinburgh.
http://www.aiai.ed.ac.uk

[16]
The Essence of Strategic Management
by C.Bowman
Published by Prentice Hall International.
1990.

[18]
Engineers in Business. The Principles of Management and Product Design
by M.Lanigan
Published by Addison-Wesley Publishers Ltd
1992

[19,62]
Foundations of Structural Optimisation: A Unified Approach.
edited by A.J.Morris
Published by John Wiley & Sons
1982

[20]
Multidisciplinary Design and Optimisation
by J.Sobieszczanski-Sobieski
AGARD Lecture Series 186
Integrated Design Analysis and Optimisation of Aircraft Structures
AGARD Paris
1992

[21]
Multidisciplinary Optimisation of a Commercial Aircraft Wing - An Exploratory Study
by C.Borland, Benon, Frank, Mastro, Barthelemy
AIAA-94-4305

[22]
Practical Architecture of Design Optimisation Software for Aircraft Structures taking the MBB Lagrange Code as an Example
by J.Krammer
AGARD Lecture Series 186
Integrated Design Analysis and Optimisation of Aircraft Structures
AGARD Paris
1992

[23]
A Multidisciplinary Approach in Computer Aided Engineering
D.J.Laan
AGARD Structures and Materials Panel Workshop on Integrated Airframe Design Technology
Sesimbra, Portugal
May 1996

[24]
A Common Framework Architecture for an Integrated Aircraft Design
J.Krammer, Vilmeier, Schuhmacher and Weber
AGARD Structures and Materials Panel Workshop on Integrated Airframe Design Technology
Sesimbra, Portugal
May 1996

[25]
BRITE/EURAM BE 95-2056

[26]
Large Scale Tracked Vehicle Concurrent Engineering Environment
K.K.Choi, Wu, Chang, Tang, Wang and Haug
in Advances in Structural Optimisation
Edited by J.Herskovits
Published by Kluwer Academic Publishers
1995

[27]
Automated Structural Analysis Process at Rockwell
S.K.Dobbs, Schwanz and Abdi
AGARD Structures and Materials Panel Workshop on Integrated Airframe Design Technology
Sesimbra, Portugal
May 1996

[28]
Integrated Airframe Design Technology
D.Thompson
AGARD Structures and Materials Panel Workshop on Integrated Airframe Design Technology
Sesimbra, Portugal
May 1996

[29]
An MDO Design Methodology for the Concurrent Aerodynamic/Cost Design of MAGLEV vehicles
by J.S.Tyll, M.A.Eaglesham, J.A.Schetz, M.Deisenroth, D.T.Mook
AIAA/NASA/ISSMO 6th Symposium on Multidisciplinary Analysis and Optimisation
Technical Papers. pt. 1
Bellevue, WA
September 4-6, 1996

[30]
A Decision-Based Approach to Concurrent Design
by F.Mistree, W.F.Smith and B.A.Bras
Handbook of Concurrent Engineering
Published by Chapman & Hall
1993

[31]
On Modelling Design Evolution Along a Design Timeline
by S.Vadde, J.K.Allen, T.Lucas and F.Mistree
AIAA/USAF/NASA/ISSMO 5th Symposium on Multidisciplinary Analysis and Optimisation
Technical Papers. pt. 2
Panama City Beach, FL
September 7-9, 1994

[32]
Knowledge-Based Manufacturing and Structural Design for a High Speed Civil Transport
by D.P.Schrage, D.N.Mavris and W.J.Marx
1st Industry/Academy Symposium on Research for Future Supersonic and Hypersonic Vehicles
Greensboro, NC
December 1994

[33]
Implementing an IPPD Environment from a Decision-Based Design Perspective
by M.A.Hale, J.I.Craig, F.Mistree, D.P.Schrage
in Multidisciplinary Design Optimisation - State of the Art;
Proceedings of the ICASE/NASA Langley Workshop
Hampton, VA

March 13-16, 1995

[34]
Decisions with Multiple Objectives Preferences and Value Tradeoffs
by R. L.Keeney and H.Raiffa
Published by the Cambridge University Press
1993

[35]
Technical Data Management for Collaborative Multi-discipline Optimisation
by S.Allwright
AIAA/NASA/ISSMO 6th Symposium on Multidisciplinary Analysis and Optimisation
Technical Papers. pt. 2
Bellevue, WA
September 4-6, 1996

[36,38]
Artificial Intelligence and The Design of Expert Systems
by G.F.Luger and W.A.Stubblefield
Published by the Benjamin/Cummings Publishing Company, Inc.
Cal., USA
1989

[37,46]
Rule-Based Expert Systems
by B.G.Buchanan and E.H.Shortliffe
Published by Addison-Wesley Publishing Company
USA
1985

[39]
Using Experience in Learning and Problem Solving
Ph.D. diss.
by P.Koton
Department of Computer Science
MIT
1988

[40]
Inductive Learning in a Mixed Paradigm Setting
by D.B.Skalak and E.L.Rissland
in Proceedings of AAAI-90.
Published by AAAI Press/MIT Press
Cambridge
1990

[41]
Case Adaptation in AutoClave Layout Design
by R.Barletta and D.Hennessy
in Proceedings: Workshop on Case-based Reasoning (DARPA),
Published by Morgan Kaufmann
Pensacola Beach, Florida. San Mateo, CA
1989

[42]
The Roles of Adaptation in Case-based Design
by T.R.Hinrichs and J.Kolodner
in Proceedings of AAAI-91
Published by AAAI Press/MIT Press
Cambridge, MA
1991

[43]
HI-RISE : An Expert System for Preliminary Structural Design
by M.L.Maher
in, Expert Systems for Engineering Design
Edited by M.D.Rychener
Academic Press, Inc., London
1988

[44]
Mesh Generation Expert System for Engineering Analyses with FEM
by Dolsak, Bojan and A.Jezernik
Computers in Industry Vol. 17 No. 2
November 1991

[45]
A Knowledge-based Consultant for Structural Analysis
by J.Bennett, L.Creary, R.Engelmore, and R.Melosh
Computer Science Department, Stanford University
Stanford, California, USA.
September 1978

[47]
A general Expert System for the Design of Capital Goods
by R.Pena
in Operational Expert System Applications in Mexico
edited by F.J.Cantu-Ortiz
Published by Pergamon Press
1991

[48]
Expert Systems for Structural Design
by H.Adelli and K.V.Balasubramanyam
Published by Prentice Hall, New Jersey, USA

1988

[49]
An Application of Expert System Techniques to Structural Optimisation in a FORTRAN Environment
RAE/TM/MAT/Str/1147
1990

[50]
Application of Artificial Neural Networks to the Design Optimisation of Aerospace Structural Component.
by L.Berke, S.N.Patnaik and P.L.N.Murthy
NASA Technical Memorandum 4389
1993

[51]
Towards a Case-based Tool for Aiding Conceptual Design Problem Solving
by A.K.Goel, J.L.Kolodner, M.Pearce, R.Billington and C.Zimring
in Proceedings: Workshop on Case-based Reasoning (DARPA), Washington, D.C.
Published by Morgan Kaufmann
San Mateo, CA
1991

[52]
Automated Reuse of Design Plans
by J.Mostow and M.Barley
in Proceedings of the 1987 International Conference on Engineering Design (ICED87),
vol. 2
Published by ASME
Boston
1987

[53]
Behavioural Synthesis in CADET, a Case-based Design Tool
by D.Navinchandra, K.Sycara and S.Narasimhan
in Proceedings of the Seventh IEEE Conference ob AI Applications
Published by IEEE Press, New York
Miami
1991

[54]
CADSYN: Using Case and Decomposition Knowledge for Design Synthesis
by M.L.Maher and D.M.Zhang
in Artificial Intelligence in Design 1991,
edited by J.S.Gero
Published by Butterworth-Heineman, Oxford
1991

[55]
Case-based Design: A Task Analysis
by A.Goel and B.Chandrasekaran
in Artificial Intelligence Approaches to Engineering Design
vol. 2: Innovative Design
edited by C.Tong and D.Sriram
Published by Academic Press, San Diego
1992

[56]
ASKJEF: Integrating Case-based Reasoning and Multimedia Technologies for Interface Design Support
by J.Barber, S.Bhatta, A.Goel, M.Jacobsen, M.Pearce, L.Penberthy, M.Shanker and E.Stroulia
in Artificial Intelligence in Design 1992
edited by J.S.Gero
Published by Kluwer Academic, Dordrecht
1992

[57]
A Case-based Design Aid for Architecture
by E.A.Domeshek and J.L.Kolodner
in Artificial Intelligence in Design 1992
edited by J.S.Gero
Published by Kluwer Academic, Dordrecht
1992

[58]
A Multimedia Approach to Case-based Structural Design
by M.L.Maher and M.B.Balachandran
Journal of Computing in Civil Engineering
No.8, vol. 3, pages 137-150
Published by ASCE
1994

[59]
http://www.catia.ibm.com/contact/contact.html

[60,61]
The MacNeal-Schwendler Corporation
815 Colorado Boulevard
Los Angeles, CA 90041-1777
USA

[63]
Introduction to Optimum Design.
by J.S.Arora
Published by McGraw-Hill, Inc.
1989

[64]
Numerical Optimisation Techniques for Engineering Design: With Applications
by G.N.Vanderplaats
Published by McGraw-Hill, Inc.
1984

[65,66]
Shaft Design Using an Advanced CAD System
by R.M.Trebilcock
MSc Thesis
Cranfield University
1994

[67,68,69]
Knowledge-based Optimum Design
by M.Balachandran
Published by Addison-Wesley Publishers Ltd.
1992

[70]
Building Knowledge-based System Towards a Methodology
by J.S.Edwards
Published by Pitman Publishing
1991

[71,74,75]
A Framework and an Analysis of Current Proposals for the Case-Based Organisation and Representation of Procedural Knowledge
by R.Zito-Wolf and R.Alterman
Proceedings of the 11th National Conference on Artificial Intelligence.
American Association for Artificial Intelligence.
Washington D.C, U.S.A.
1993

[72]
Expert System for Structural Optimisation Exploiting Past Experience
by W.Kuntjoro
Ph.d Thesis
College of Aeronautics
Cranfield University
1994

[73]
Case-Based Reasoning, Constraints, and Object-Oriented Programming for Material Expert Systems
by K.J.Meltsner
in Knowledge-Based Applications in Materials Science and Engineering
edited by J.K.McDowell & K.J.Meltsner
The Minerals, Metals & Materials Society

1994

[76]
Encyclopaedia of Artificial Intelligence
Volume 2
edited by S.C.Shapiro and D.Eckroth
Published by John Wiley and Sons, (New York)
1989

[77]
Genetic Algorithms in Search, Optimisation and Machine Learning
by D.E.Goldberg
Published by Addison Wesley
1989

[78,79]
A Role for Genetic Algorithms in a Preliminary Design Environment
by P.Gage and I.Kroo
Aircraft Design, Systems and Operations Meeting (August 11th - 13th, 1993)
Published by the American Institute of Aeronautics and Astronautics
1993

[80]
Design Optimisation using the BEM Coupled with Genetic Algorithm
by J.A.Vasconcelos, L.Krahenbuhl, L.Nicolas; A.Nicolas
Second International Conference on Computation in Electromagnetics
Published by IEE, UK.
1994

[81]
Application of Artificial Neural Networks to the Design Optimisation of Aerospace Structural Components
by L.Berke, S.N.Patnaik, P.L.N.Murthy
NASA Technical Memorandum 4389
Published by the National Aeronautics and Space Administration
1993

[82]
A Practical Guide to Neural Nets
by M.McCordNelson and W.T.Illingworth
Published by Addison Wesley
1991

[83]
Introduction to Neural Networks
by P.Picton
Published by Macmillian Press Ltd
1994

[84]
Software Engineering (fourth edition)
by Ian Sommerville
Published by Addison-Wesley Publishers Ltd
1992

[85]
Concurrent Engineering (Final Year Project)
by R.M.Trebilcock
University of Salford
1993

[86]
CBR in Battle Planning.
by M.Goodman.
Proceedings: Workshop on case-based reasoning (DARPA),
Pensacola Beach, Florida.
1989

[87]
Representing Cases as Knowledge Sources that Apply Local Similarity Metrics.
by D.B.Shalak.
Fourteenth Annual Conference of the Cognitive Science Society.
Northvale
Published by Erlbaum.
1992

[88]
CHEF: A Model of Case-based Planning
by K.Hammond.
Proceedings: AAAI-86
Cambridge, MA.
Published by AAAI Press/MIT Press
1986

[89]
Bayesian Theory
by J.M.Bernardo and A.F.M.Smith
Published by John Wiley & Sons Ltd.
1994

[90]
Algebraic Simplification
by D.McAllester
In Proceedings of IJC AI-81
Published by Morgan Kaufmann,
Palo Alto, California. USA
1987

[91]
Diagnostic Reasoning Based on Structure and Behaviour
by R.Davis
Artificial Intelligence
Volume 24
1984

[92]
Diagnosing Circuits with State: An Inherently Underconstrained Problem
In Proceedings of AAAI-84
Published by Morgan Kaufman,
Palo Alto, California
1984

[93]
Order-of-magnitude Reasoning with O[M]
by M.L.Mavrovouniotis and G.Stephanopoulos
Artificial Intelligence in Engineering
Volume 3, No. 3
1989

[94,95]
Case-Based Reasoning
by J.Kolodner
Published by Morgan Kaufmann Publishers, Inc.
1993

[96]
http://www.ibm.com

[97]
Concentra
Viscount Centre 11
Milburn Hill Road,
University of Warwick,
Science Park,
Coventry CV4 7HS

[98]
Object-Oriented Programming in C++
Second Edition
by Robert Lafore
Published by the Waite Group Press
1995

[99]
Borland Paradox
for Windows 95 and Windows NT,
Version 7.
Borland International, Inc.
1996.

[100,102]
Teach Yourself Borland C++ Builder in 21 Days
by K.Reisdorph and K.Henderson
Published by Sams Publishing
1997

[101]
Visual Component Library Reference
Volume 1 and 2
Borland C++ Builder for Windows 95 and Windows NT
Published by Borland Internationl
1997

[103]
POET Software GmbH
Kattjahren 4-8
22359 Hamburg,
Germany

# Bibliography


Structural Optimisation Short Course Notes
by A.J.Morris
College of Aeronautics
Cranfield University
1995

Handbook of Genetic Algorithms
edited by L.Davis
Published by Van Nostrand Reinhold
1991.

Knowledge Based Engineering
Jon Boyce
Computervision.
Published by The Computervision (UK) Users Group.
1990.

Expert Systems Design and Development
by J.Durkin
Published by Macmillan Publishing Company
1994

Artificial Intelligence, A knowledge-Based Approach
by M.W.Firebaugh
Published by PWS-KENT Published Company
1989

Expert Systems Architectures
by L.Johnson and E.T.Keravnou
Published by Kogan Page Limited.
1988.

Data Structures - An Object Oriented Approach
by William J.Collins
Published by Addison-Wesley Publishing Company, Inc.
1992

Developing Object Oriented Data Structures Using C++
by Alistair McMonnies and W.Stewart McSporran
Published by McGraw-Hill International (UK) Limited
1995

S83 V/STOL Fighter Foreplane Design
by N.G.Lillywhite.
MSc Thesis 1984
College of Aeronautics, Cranfield University.

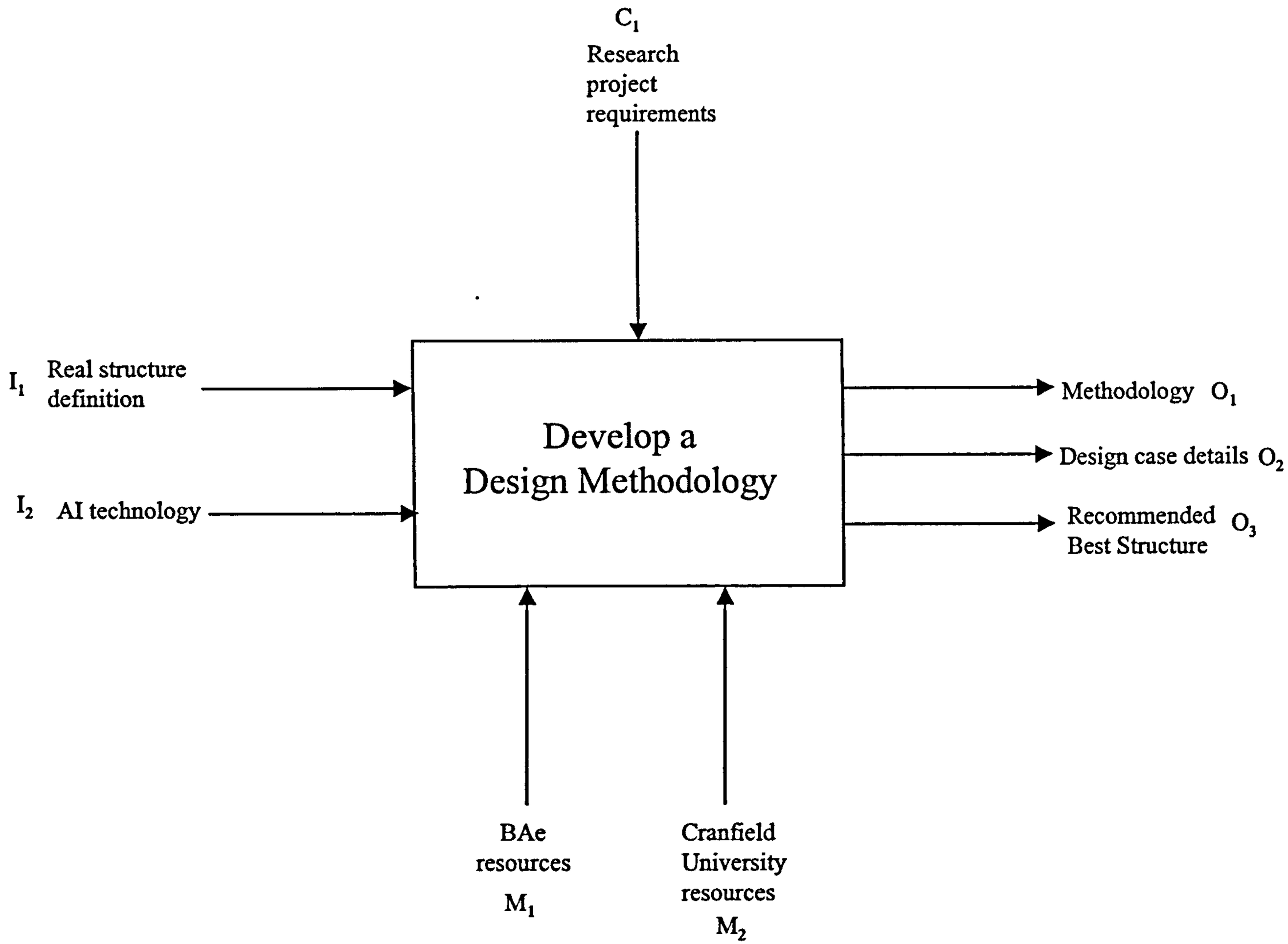
Object-Oriented Design with Applications
by Grady Booch
Published by The Benjamin/Cummings Publishing Company Inc.
1991

TF-89 Tactical Fighter Forplane Design
by Jean-Francois Galopin
MSc Thesis 1990
College of Aeronautics, Cranfield University.

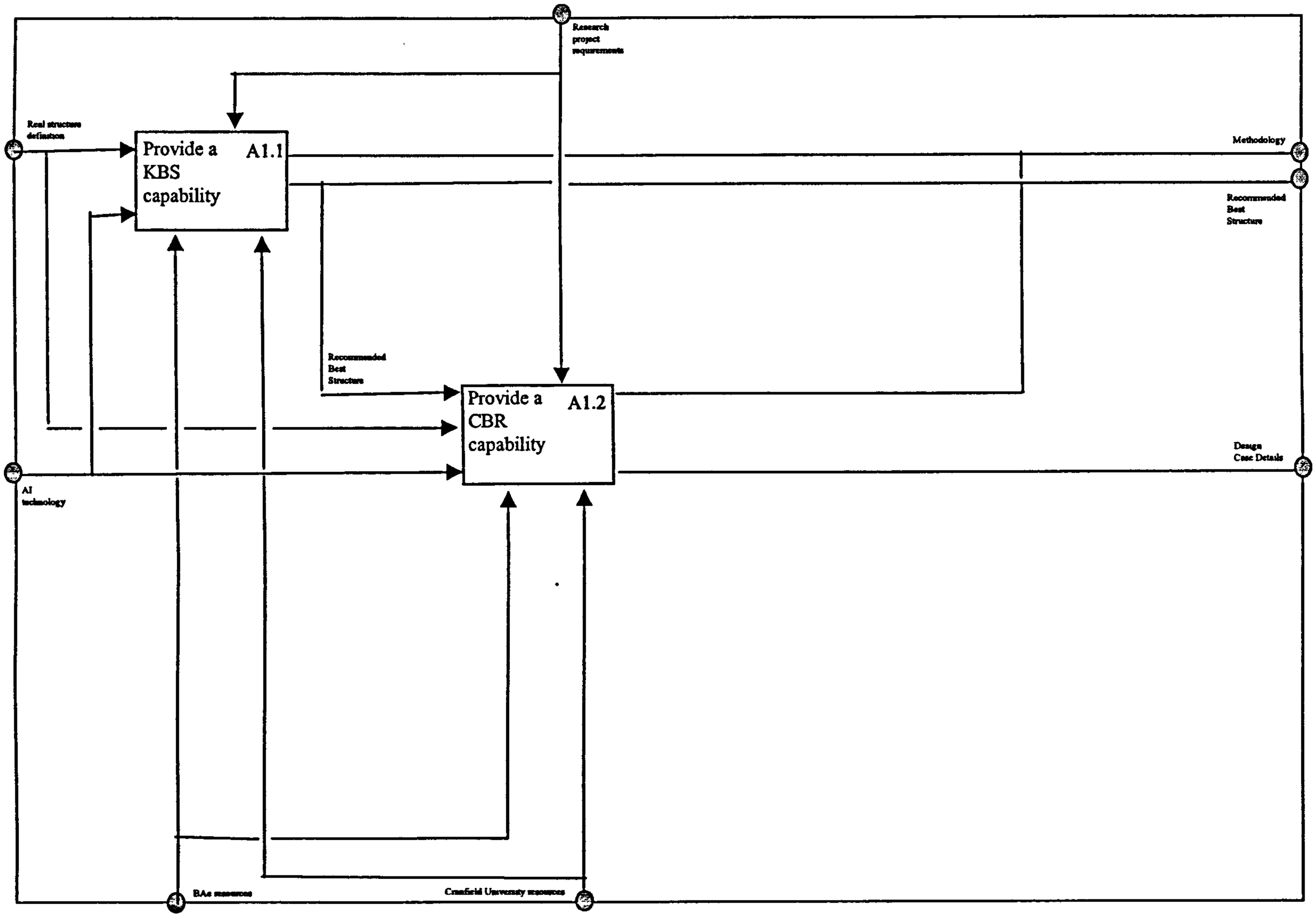
Composite Airframe Structures - Practical Design Information and Data
by Michael C.Y.Niu
Published by Conmilit Press Ltd
1992

# Appendix A

# IDEF0 System Analysis of the ICES Methodology



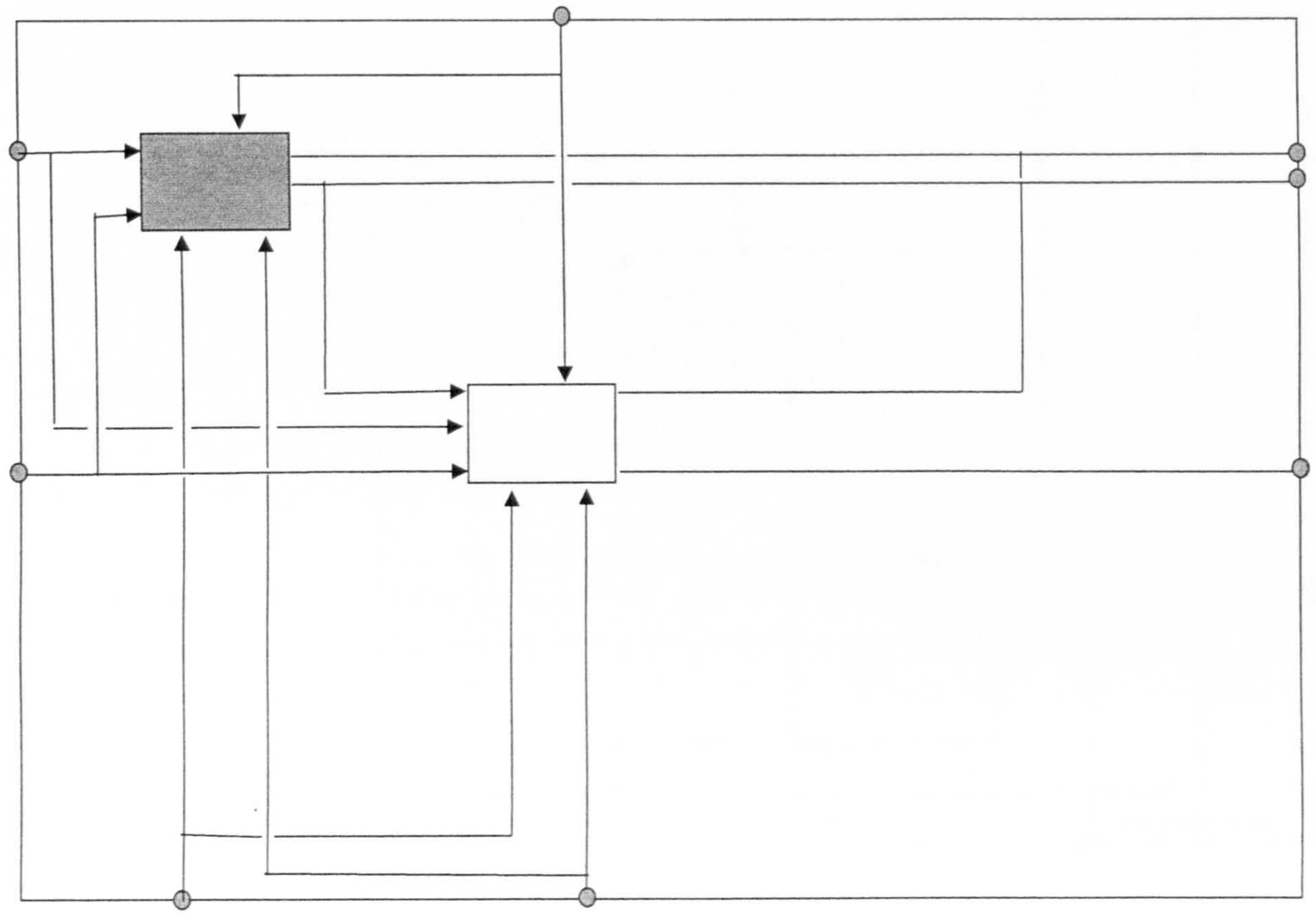

Context Diagram: 'Develop a Design Methodology'

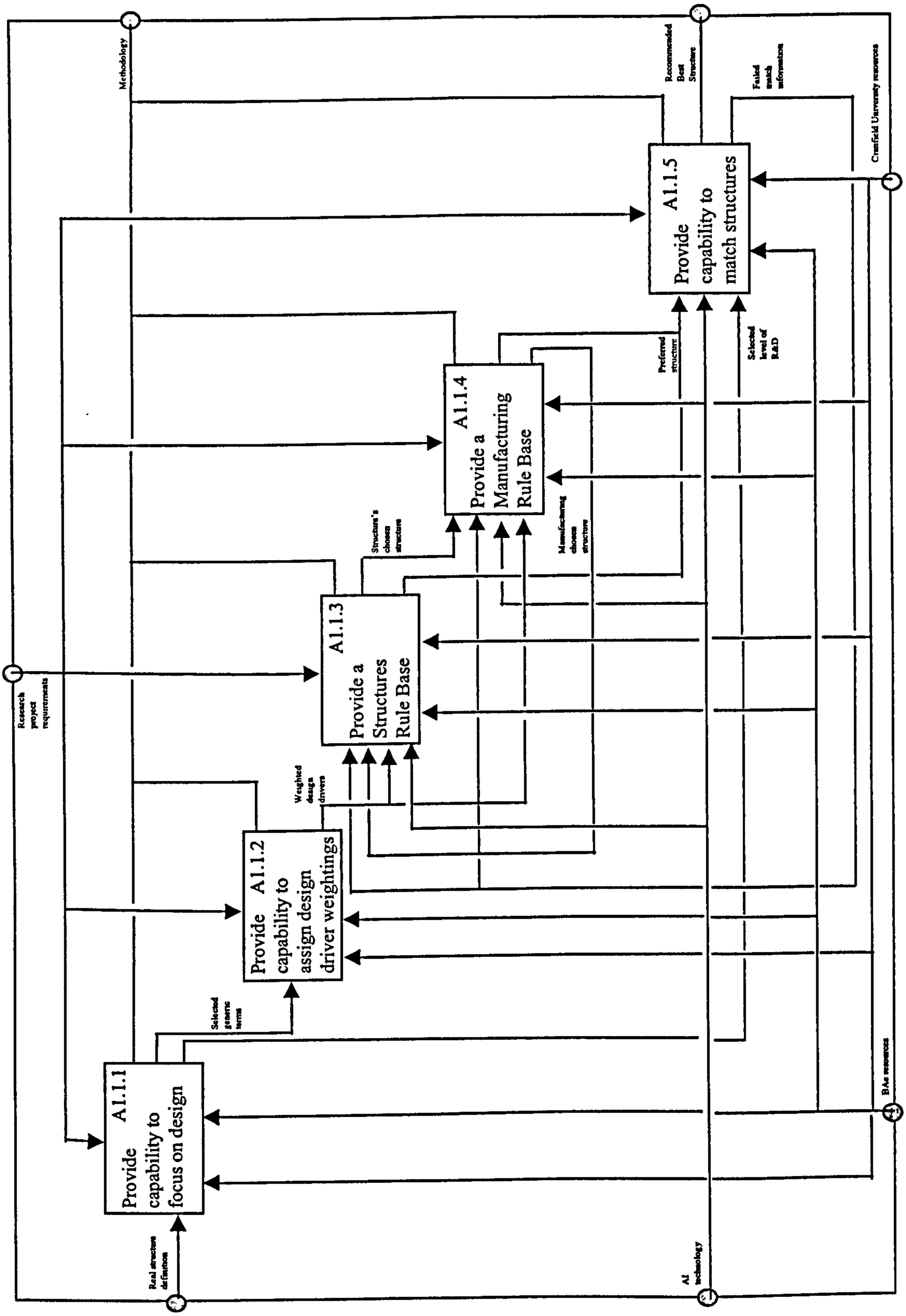



## A0: 'Develop a Design Methodology'

A1.1
Knowledge-Based System

A1.2
Case-Based Reasoning System

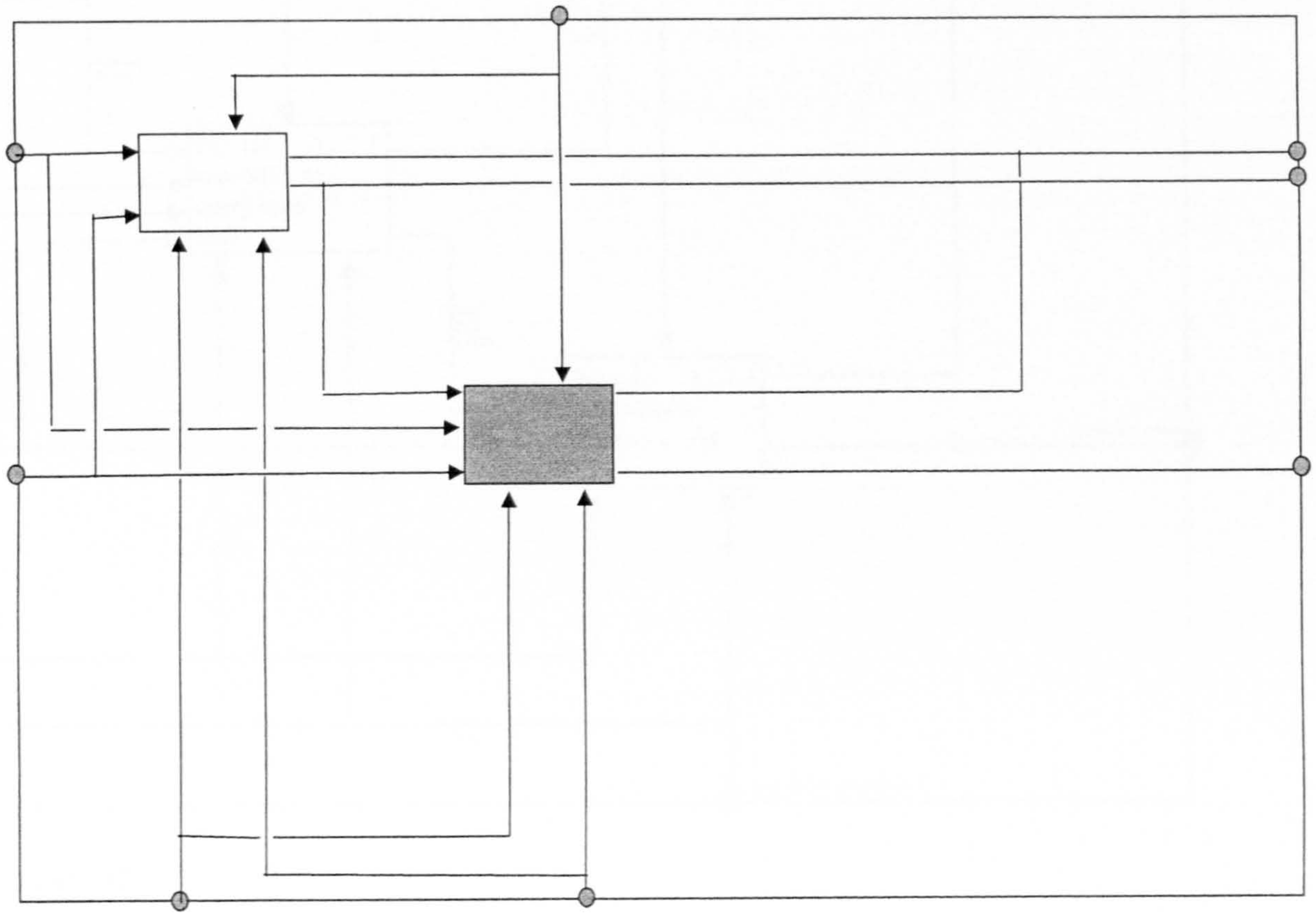


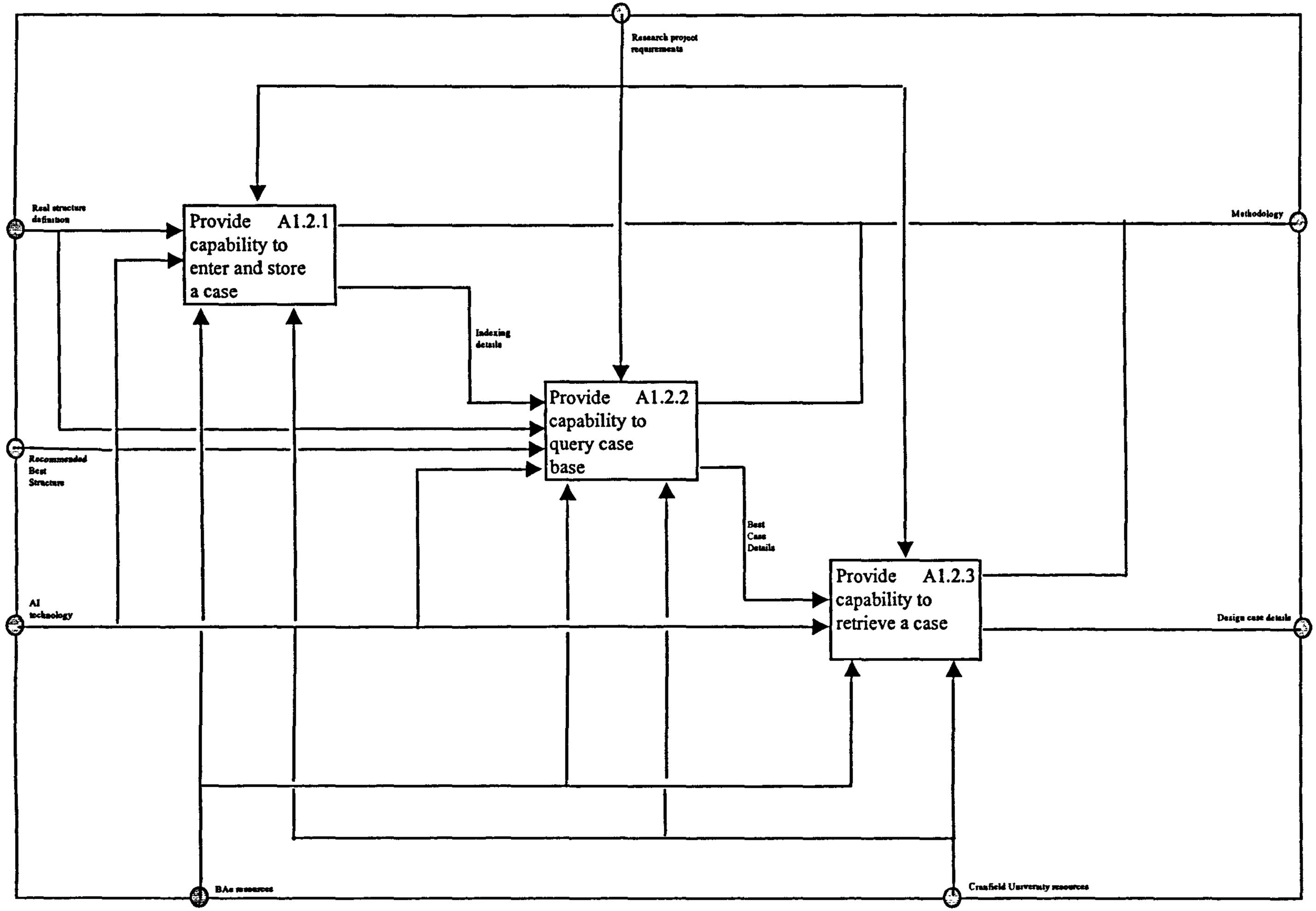



**A1.1: 'Provide a KBS Capability'**

A1.1
Knowledge-Based System

A1.2
Case-Based Reasoning System






## A1.2: 'Provide a CBR Capability'

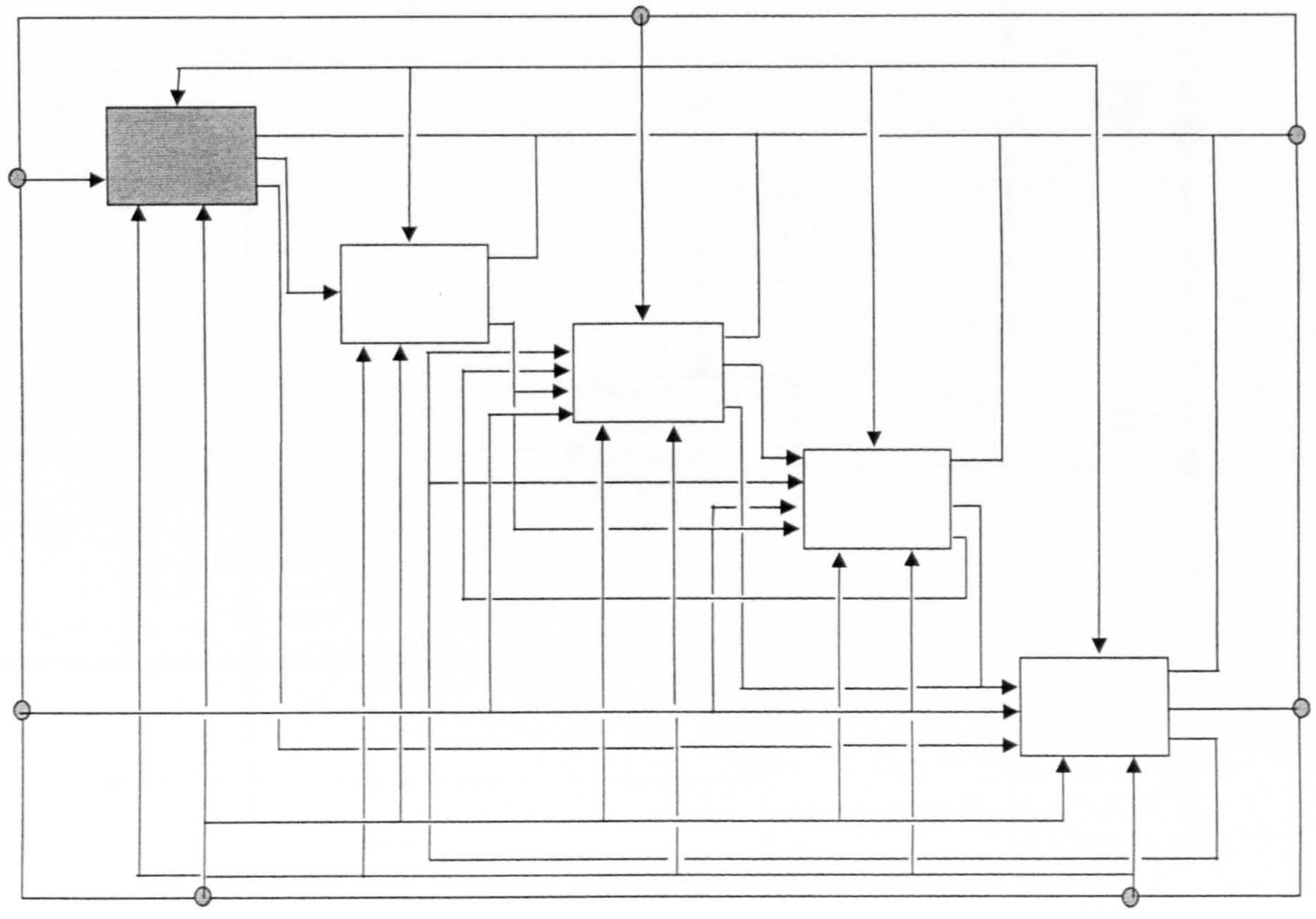
A1.1.1
Focus on design

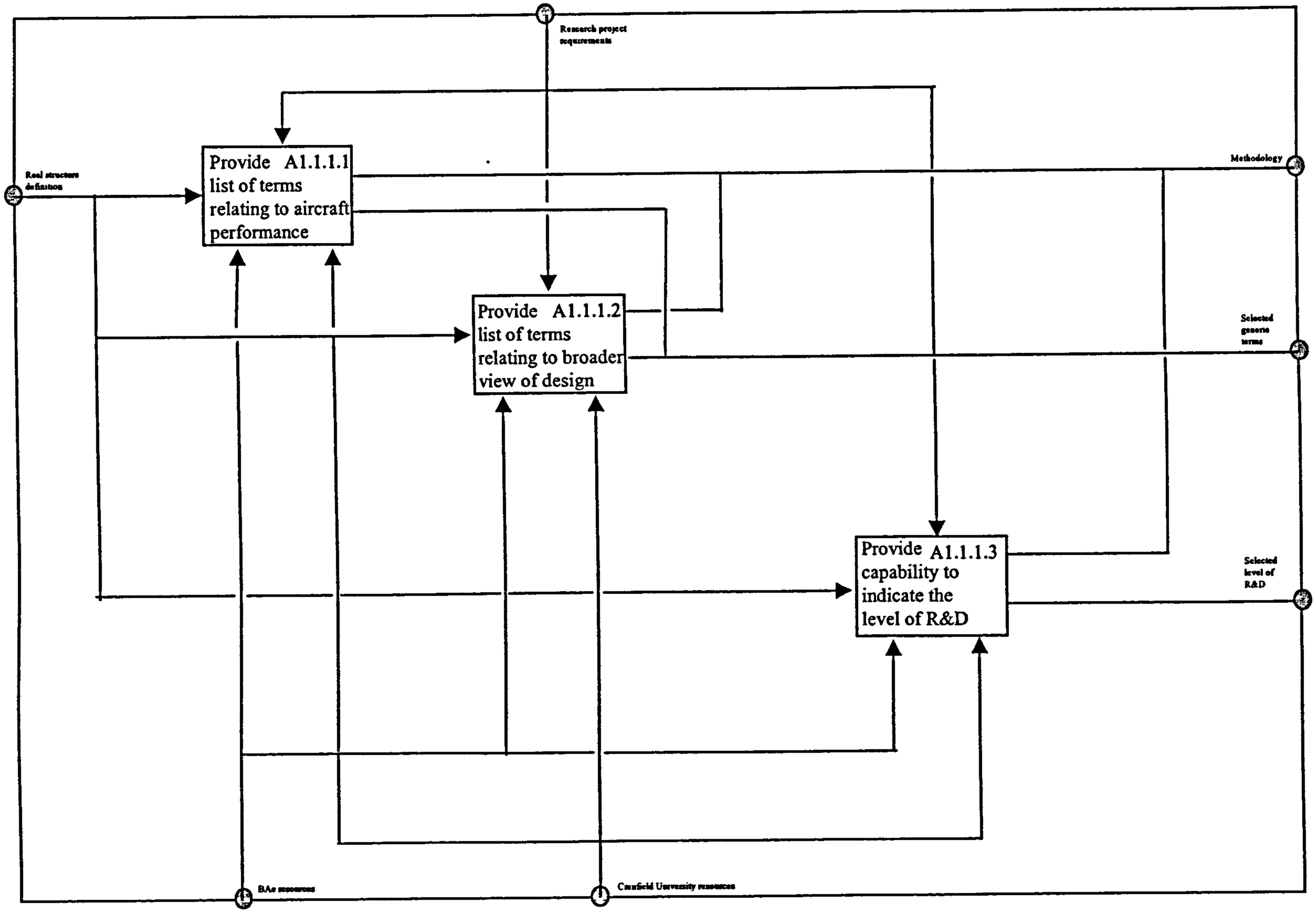
A1.1.2
Assign design driver weightings

A1.1.3
Structures rule base

A1.1.4
Manufacturing rule base

A1.1.5
Match structural preferences






## A1.1.1: 'Provide Capability to Focus on Design'

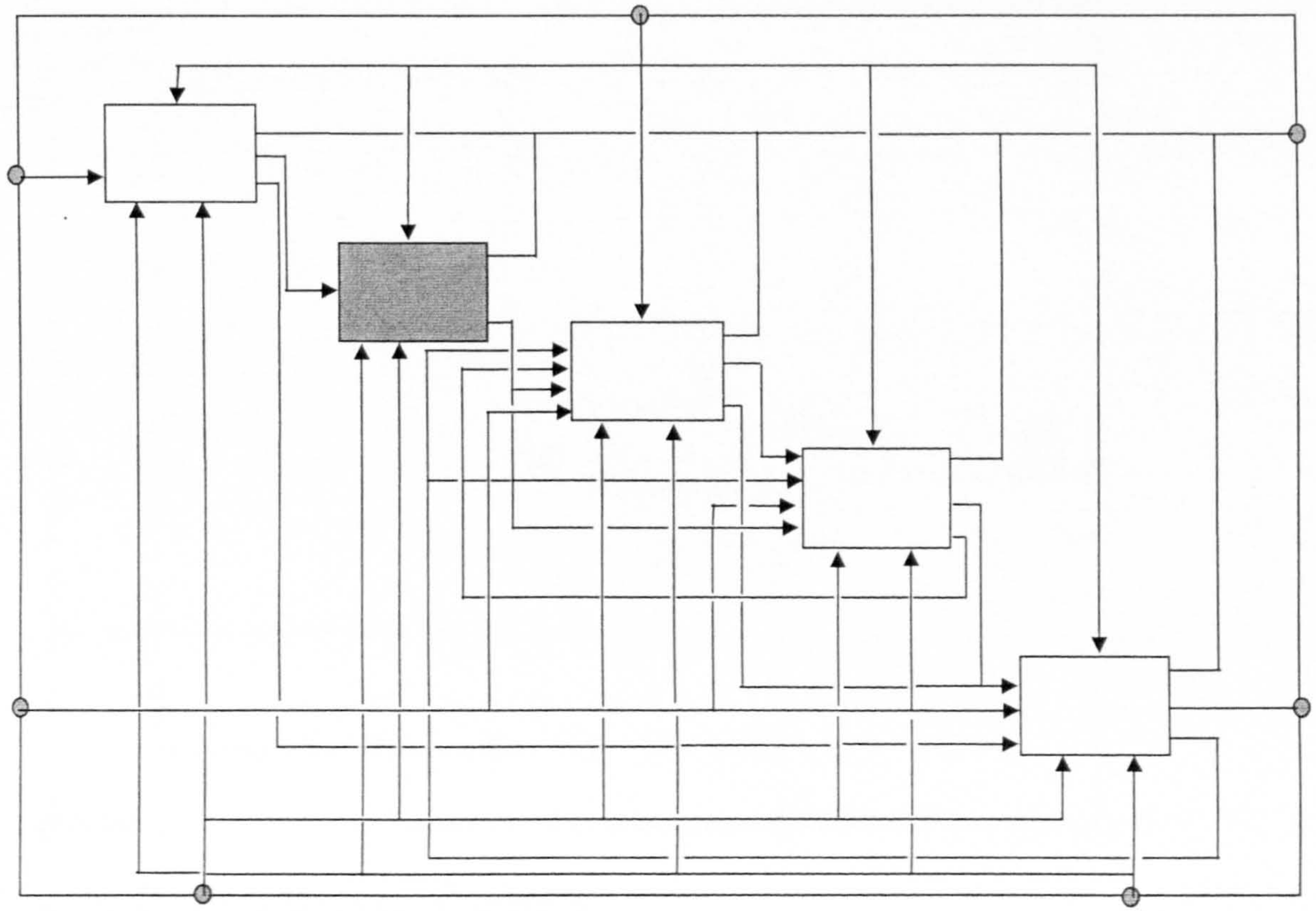
A1.1.1
Focus on design

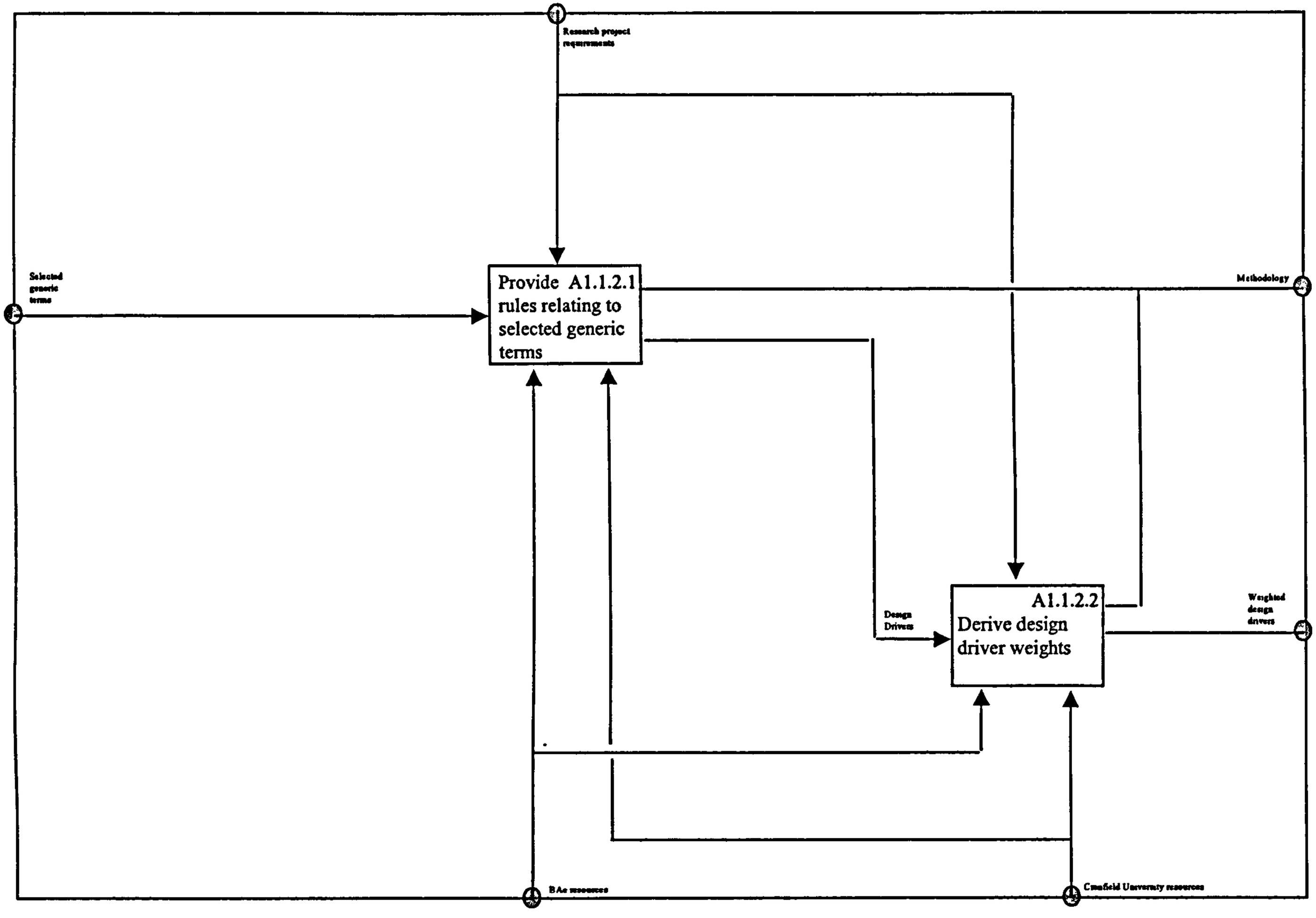
A1.1.2
Assign design driver weightings

A1.1.3
Structures rule base

A1.1.4
Manufacturing rule base

A1.1.5
Match structural preferences

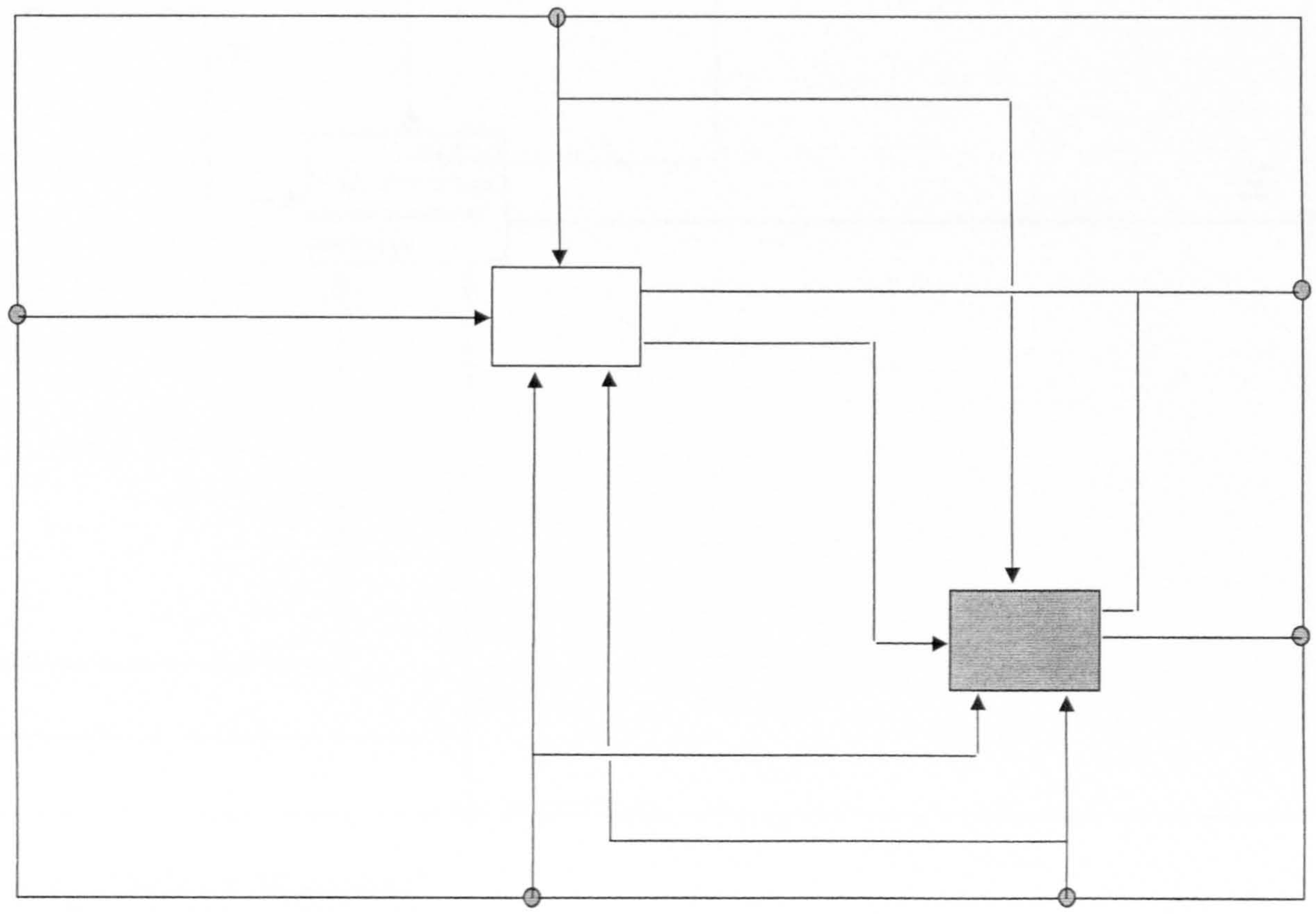


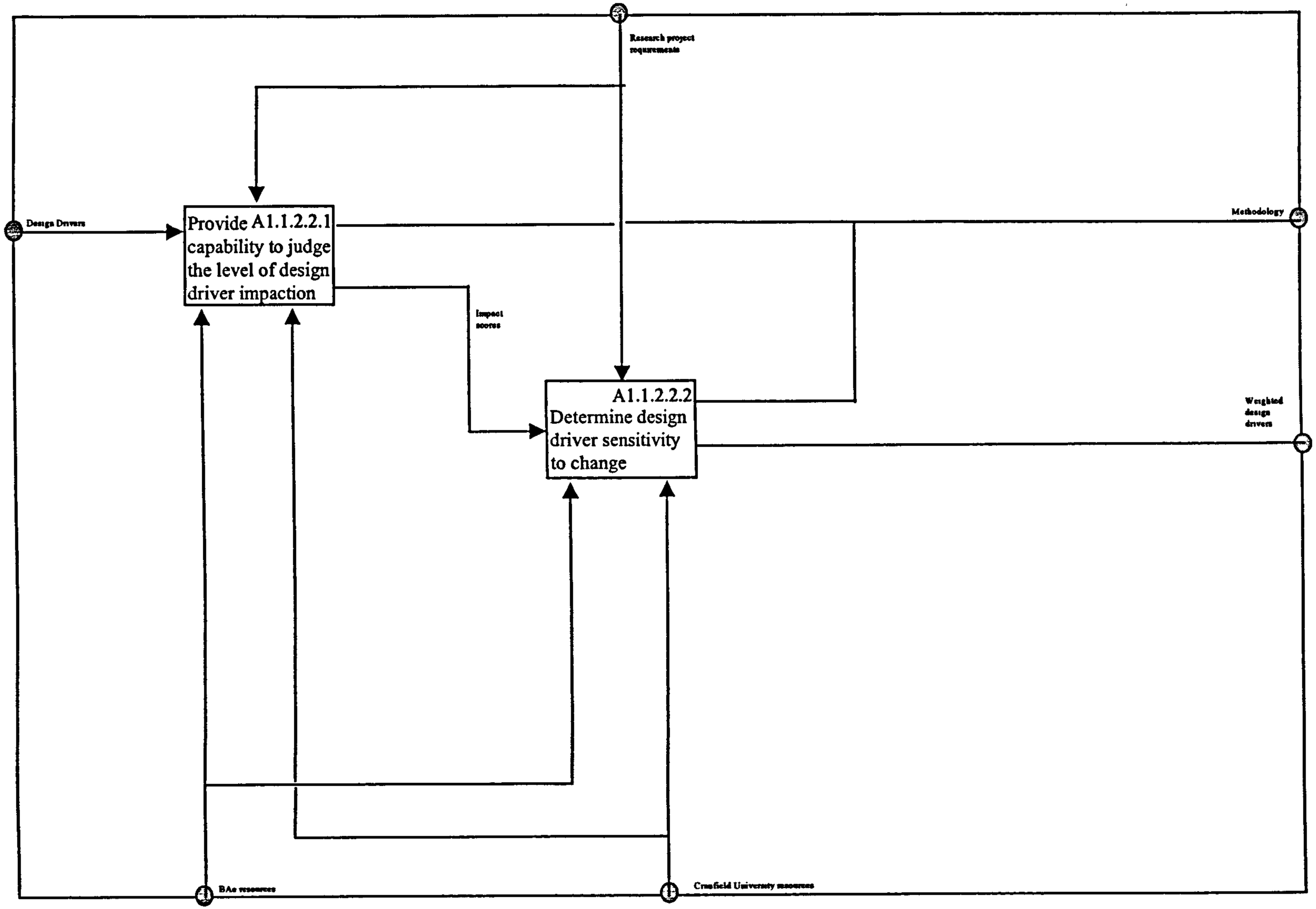



## A1.1.2: 'Provide Capability to Assign Design Driver Weights'

A1.1.2.1
Rules relating to generic terms

A1.1.2.2
Design driver weights






## A1.1.2.2: 'Derive Design Driver Weights'

# Data Dictionary ICOM

## A0: Develop a Design Methodology

| Name | Type | Where from | Where to |
|---|---|---|---|
| | | | |
| Real Structure Definition | Input | Environment | A1.1, A1.2 |
| | | | |
| Research Project Requirements | Control | Environment | A1.1, A1.2 |
| | | | |
| Methodology | Output | A1.1, A1.2 | Environment |
| | | | |
| Recommended Best Structure | Output, Input | A1.1 | Environment, A1.2 |
| | | | |
| Design Case Details | Output | A1.2 | Environment |
| | | | |
| Cranfield University | Mechanism | Environment | A1.1, A1.2 |
| | | | |
| BAe Resouces | Mechanism | Environment | A1.1, A1.2 |
| | | | |
| AI Technology | Input | Environment | A1.2 |

## A1.1: Provide a KBS Capability

| Name | Type | Where from | Where to |
|---|---|---|---|
| | | | |
| Real Structure Definition | Input | A0 | A1.1.1 |
| | | | |
| Research Project Requirements | Control | A0 | A1.1.1, A1.1.2, A1.1.3, A1.1.4, A1.1.5 |
| | | | |
| Methodology | Output | A1.1.1, A1.1.2, A1.1.3, A1.1.4, A1.1.5 | A0 |
| | | | |
| Recommended Best Structure | Output | A1.1.5 | A0 |
| | | | |

| | | | |
|---|---|---|---|
| Cranfield University | Mechanism | A0 | A1.1.1, A1.1.2, A1.1.3, A1.1.4, A1.1.5 |
| | | | |
| BAe Resources | Mechanism | A0 | A1.1.1, A1.1.2, A1.1.3, A1.1.4, A1.1.5 |
| | | | |
| AI Technology | Input | A0 | A1.1.3, A1.1.4, A1.1.5 |
| | | | |
| Selected Design Drivers | Output, Input | A1.1.1 | A1.1.2 |
| | | | |
| Weighted Design Drivers | Output, Input | A1.1.2 | A1.1.3, A1.1.4 |
| | | | |
| Structure's Chosen Structure | Output, Input | A1.1.3 | A1.1.4 |
| | | | |
| Manufacturing Chosen Structure | Output, Input | A1.1.4 | A1.1.3 |
| | | | |
| Preferred Structure | Output, Input | A1.1.3, A1.1.4 | A1.1.5 |
| | | | |
| Selected level of R&D | Output, Input | A1.1.1 | A1.1.5 |
| | | | |
| Failed Match Information | Output, | A1.1.5 | A1.1.3, A1.1.4 |

**A1.1.1: Provide Capability to Focus on Design**

| Name | Type | Where from | Where to |
|---|---|---|---|
| | | | |
| Real Structure Definition | Input | A1.1 | A1.1.1.1, A1.1.1.2 A1.1.1.3 |
| | | | |
| Research Project Requirements | Control | A1.1 | A1.1.1.1, A1.1.1.2 A1.1.1.3 |
| | | | |
| Methodology | Output | A1.1.1.1, A1.1.1.2 A1.1.1.3 | A1.1 |
| | | | |
| Selected Generic Terms | Output | A1.1.1.1, A1.1.1.2 | A1.1 |

| | | | |
|---|---|---|---|
| Selected level of R&D | Output | A1.1.1.3 | A1.1 |
| | | | |
| Cranfield University Resources | Mechanism | A1.1 | A1.1.1.1, A1.1.1.2 A1.1.1.3 |
| | | | |
| BAe Resources | Mechanism | A1.1 | A1.1.1.1, A1.1.1.2 A1.1.1.3 |

**A1.1.2: Provide Capability to Assign Design Driver Weights**

| Name | Type | Where from | Where to |
|---|---|---|---|
| | | | |
| Selected Generic Terms | Input | A1.1 | A1.1.2.1 |
| | | | |
| Research Project Requirements | Control | A1.1 | A1.1.2.1, A1.1.2.2 |
| | | | |
| Methodology | Output | A1.1.2.1, A1.1.2.2 | A1.1 |
| | | | |
| Cranfield University Resources | Mechanism | A1.1 | A1.1.2.1, A1.1.2.2 |
| | | | |
| BAe Resources | Mechanism | A1.1 | A1.1.2.1, A1.1.2.2 |
| | | | |
| Design Drivers | Output, Input | A1.1.2.1 | A1.1.2.2 |

**A1.1.2.2: Derive Design Driver Weights**

| Name | Type | Where from | Where to |
|---|---|---|---|
| | | | |
| Design Drivers | Input | A1.1.2 | A1.1.2.2.1 |
| | | | |
| Research Project Requirements | Control | A1.1.2 | A1.1.2.2.1 A1.1.2.2.2 |
| | | | |
| Methodology | Output | A1.1.2.2.1 A1.1.2.2.2 | A1.1.2 |
| | | | |
| Weighted Design Drivers | Output | A1.1.2.2.2 | A1.1.2 |

| | | | |
|---|---|---|---|
| Cranfield University Resources | Mechanism | A1.1.2 | A1.1.2.2.1<br>A1.1.2.2.2 |
| | | | |
| BAe Resources | Mechanism | A1.1.2 | A1.1.2.2.1<br>A1.1.2.2.2 |
| | | | |
| Impact Scores | Output, Input | A1.1.2.2.1 | A1.1.2.2.2 |
| | | | |

**A1.2: Provide a CBR Capability**

| Name | Type | Where from | Where to |
|---|---|---|---|
| | | | |
| Real Structure Definition | Input | A0 | A1.2.1, A1.2.2 |
| | | | |
| Research Project Requirements | Control | A0 | A1.2.1, A1.2.2, A1.2.3 |
| | | | |
| Methodology | Output | A1.2.1, A1.2.2, A1.2.3 | A0 |
| | | | |
| Design Case Details | Output | A1.2.3 | A0 |
| | | | |
| Cranfield University Resources | Mechanism | A0 | A1.2.1, A1.2.2, A1.2.3 |
| | | | |
| BAe Resources | Mechanism | A0 | A1.2.1, A1.2.2, A1.2.3 |
| | | | |
| AI Technology | Input | A0 | A1.2.1, A1.2.2, A1.2.3 |
| | | | |
| Recommended Best Structure | Input | A0 | A1.2.2 |
| | | | |
| Indexing Details | Output, Input | A1.2.1 | A1.2.2 |
| | | | |
| Best Case Details | Output, Input | A1.2.2 | A1.2.3 |

## Node List

A0 Develop a Design Methodology
A1.1 Provide a KBS capability
A1.1.1 Provide capability to focus on design
A1.1.1.1 Provide list of terms relating to aircraft performance
A1.1.1.2 Provide list of terms relating to broader view of design
A1.1.1.3 Provide capability to indicate the level of R&D
A1.1.2 Provide capability to assign design driver weights
A1.1.2.1 Provide rules relating to selected generic terms
A1.1.2.2 Derive design driver weights
A1.1.2.2.1 Provide capability to judge the level of design driver impaction
A1.1.2.2.2 Determine design driver sensitivity to change
A1.1.3 Provide a Structures Rule Base
A1.1.4 Provide a Manufacturing Rule Base
A1.1.5 Provide capability to match structures
A1.2 Provide a CBR capability
A1.2.1 Provide capability to enter and store a case
A1.2.2 Provide capability to query case base
A1.2.3 Provide capability to retrieve a case

# Appendix B

# First ICES

# Prototype Documentation

# Appendix B - First ICES Prototype Documentation

This appendix contains the supporting documentation for the first ICES prototype. The documentation states what level of performance the user can expect from the software. In addition, the documentation provides a step-through example of the KBS and case base operation.

## B.1, Introduction - The Purpose and Aims of the First ICES Prototype (Prototype1)

It is important to note that the purpose of a prototype is significantly different from a fully developed system, the two should not be confused. The under-lying aim of the prototype is to demonstrate the functionality of the proposed system and the potential that exists for a fully developed system. A prototype, while focusing on functionality does not attempt to present the user with the level of capability and user-friendliness that one might expect from a fully developed system. The principle aim of the prototype is to provide interested parties with a focal point for constructive criticism and no more. The remainder of this report now provides a description of Prototype 1.

## B.2, Prototype 1 - General Description, Mode of Operation

The purpose of Prototype 1 is to identify the 'best structure' and provide the designer with efficient and effective assistance at the conceptual design stage. Prototype 1 has an integral knowledge-based system (KBS) which is closely linked to a case base. This enables heuristic knowledge i.e., rules-of-thumb, and previous design information to be made readily available to assist with the current design problem. In functional terms, Prototype 1 demonstrates the operation of both a KBS and a case base and the possibility for interaction between the two. The remainder of this section will now outline the operation of Prototype 1.

### B.2.1. Main Menu

On starting Prototype 1 the user is immediately confronted with the Main Menu. This offers the following three options, any one of which may be selected:

1, KBS Operation.
2, Case Base Operation.
3, Quit.

#### B.2.1.1. KBS Operation

The KBS Operation menu option is aimed at the user who knows what he or she wants to design but is uncertain with respect to which structure is most suitable. On selection of the KBS Operation menu option the user is presented with a menu which prompts the user to select any one of a series of terms. These terms relate to what the structure under consideration provides for the aircraft and to the broader company view of design. On the selection of these terms the system will ask the user various questions concerning the design under consideration. The answers provided to these questions generate numerically weighted constraints that consequently act as the design drivers on the process.

Having answered all the questions presented by the KBS, the system then presents the user with the KBS Menu with the following options:

1. KBS Output.
2. Show Current Constraints.
3. Revise Current Constraints.
4. Repeat KBS Questions.
5. Show Default Constraints.
6. Return to Main Menu.

**B.2.1.1.1. KBS Output**

On selection of the KBS Output menu option from the KBS Operation menu one of two mutually exclusive outputs are possible. However, it is first necessary to provide some additional insight into the operation of the KBS in order to explain how these two outputs are derived.

The KBS is in fact a super-set of two smaller rule bases; a structures rule base and a manufacturing rule base. Taking on board the constraints outlined in section B.2.1.1, each rule base works through each structural alternative in turn i.e., super plastic formed diffusion bonded (SPF/DB) titanium structures, carbon fibre composite (CFC) structures and traditional stressed skin structures, and determines which structure most satisfactorily meets the constraints from a manufacturing and a structural point of view. The preferred structure from the manufacturing and the structures rule bases are then compared with each other to see if there is a match, i.e. do the two rule bases concur with respect to what is the best structure.

Returning to the two possible outputs from the KBS Output menu option indicated at the beginning of this section. The first output is the structure that the system considers to be 'best' from both a structural perspective and a manufacturing perspective. In addition, the system proceeds to offer the user the opportunity to view the reasoning that supports this selection.

The second form of output from the prototype is where the system can not identify a structure that is 'best' from both a manufacturing and structural perspective, i.e. the two rule bases are not in agreement. The system tells the user that a 'best' structure can not be identified. The system now offers the user the opportunity to view the reasoning behind the structures and manufacturing rule bases. Having presented the user with the opportunity to view this reasoning, the system now presents the KBS Operation menu to the user. From this menu the user may select the Revise Current Constraints menu option. As the title of this menu option indicates, this option permits the user to revise the weighted values of constraints; the intention being for the user then to re-select the KBS Output menu option and see if the structures and manufacturing rule bases now agree on a preferred structure.

Clearly, it may not be possible to get an agreement between the two rules bases on what is the best structure, no matter how the constraints are altered or relaxed. It is important in this instance that the designer takes note of the explanations provided by each rule base as these explanations should provide the designer with an insight into

the reason(s) why no match between the two rule bases is possible. The user should be aware that all design aids no matter how sophisticated are only design aids and are not substitutes for the designer.

For both the first and second forms of output described above the system will also present the designer with the existing design case stored in memory which most closely corresponds to the user's input. The purpose of presenting this case to the user is that it may assist in determining which structure is most suitable for his or her needs; possibly assisting the designer to decide which constraints should be relaxed.

**B.2.1.1.2 Show Current Constraints.**
This option from the KBS Operation menu permits the user to view the numerical weightings that where attributed to each of the design constraints during the user's interaction with the system, i.e. due to the answering of prompted questions.

**B.2.1.1.3 Revise Current Constraints.**
As indicated in section B.2.1.1.1 this option enables the user to alter the values of the numerical weightings attributed to constraints. It should be noted that where a constraint is no longer to be considered in a design problem then it must be assigned a value of zero in order to eliminate it from the problem i.e., the constraint is totally relaxed.

**B.2.1.1.4 Repeat KBS Questions.**
This option permits the user to alter his or her answers to questions prompted by the system.

**B.2.1.1.5 Show Default Constraints.**
For each constraint that acts on the design process there is assigned a default numerical value. This menu option enables a list of the default design constraints to be shown. When the user answers the questions relating to the design problem he or she has the opportunity to either accept the default constraint value or enter a new value. The principle purpose of the default constraint list is to provide the user with some guidance with respect to what a suitable numerical value for a constraint might be.

**B.2.1.1.6 Return to Main Menu.**
This option permits the user to exit the KBS and return to the Main Menu. On leaving the KBS the system resets all the constraint values to their default settings.

**B.2.1.2 Case Base Operation.**
The case base is aimed at the user who has specific design specifications concerning a design problem, e.g. performance requirements, manufacturing limitations, geometric limitations. What the user wants to know is whether a design case exists in the case base that closely matches his or her requirements. If the user is interested in what

design cases exist that meet the current design specifications the Case Base Operation menu option should be selected from the Main Menu. Selection of the Case Base Operation menu option will present the following sub-menu options:

1. Enter New Case
2. Query Case Base
3. Return to Main Menu

#### B.2.1.2.1 Enter New Case

This menu option permits the user to enter a new design case(s). On the selection of this option the user is prompted by the system to enter case details. The user may enter as many cases as he or she desires. On completion of the entry of case details the case(s) are written to a file.

#### B.2.1.2.2 Query Case Base

The purpose of this menu option is to determine which design case residing in the case base most closely matches the current design problem. On the selection of this menu option the user is prompted by the system to enter the specification for the current design problem. Once the design problem specification has been entered, the system compares it with the specifications of the stored cases residing in the case base. The stored case that most closely matches the current design specification is presented to the user as being the 'best match'. The system uses the *nearest neighbour* matching approach in order to determine which case is the best matching one.

#### B.2.1.2.3 Return to Main Menu

This menu option permits the user to exit the case base and return to the main menu. On exiting the case base all the previous case query details cease to apply.

### B.3. The Level of User-Friendliness

While Prototype 1 is not 'user-unfriendly', the user-friendly capability that is present aims solely at guiding the user through the operation of the prototype and no more. The prototype operates in a Windows environment but possesses no windowing capability, menus are not of the pull-down type. Prototype 1 has an MS-DOS 'feel' to it, possessing a scrollable menuing system. Unlike a fully developed system, Prototype 1 has no help facility and as such the user should not look for one. If problems are encountered during the operation of the prototype this document should provide sufficient guidance to get the user out of trouble.

### B.4. Error Handling

Prototype 1 possesses a reasonable level of error handling. However, the error handling is not totally comprehensive but focuses on specific areas of the program's operation. Every attempt has been made to ensure that the user cannot enter contradictory information. The system will inform the user if he or she has attempted to do so e.g., if user attempts to implement constraints for batch and one-off

production at the same time the system will not accept this. During operation, Prototype 1 is required to perform a large amount file handling i.e., writing data to a file, reading data from a file, and counting data segments on a file. If any of the file handling fails during the operation of the program the user will be informed.

### B.4.1. Known Sources of Error

The principle source of error that the user is likely to encounter when operating Prototype 1 is when the user is required to input answers to prompted questions. If the user enters the wrong data type in response to a prompt the system is likely to 'crash' i.e., the input window will scroll continuously. The only method to stop this scrolling is to press the keys *Ctrl*, *Alt* and *Del* together. The user will then have to restart the program. While this source of error can readily be eliminated through effective error handling it does require a relatively high volume of code. As such, while this error handling capability will be present in the final system it was not considered essential to include it in Prototype 1. During the operation of Prototype 1, the user is advised to take care and ensure that all prompted questions are answered in the correct format.

## B.5. Prototype 1 - Operational Limitations

The following sections outline the known operational limitations of Prototype 1

### B.5.1. Capability of KBS to Determine the 'Best' Structure

In order for the KBS to derive the 'best' structure there is a requirement to have a fully developed manufacturing and structures rules bases. Unfortunately, due to time limitations it was not possible to fully develop these rule bases. For instance, Prototype 1 possesses no coded information relating to the specific manufacturing requirements of Standard x-core or Spaced x-core. As such, due to this general lack of coded information the system is unable to derive the 'best' structure. However, while acknowledging the presence of this limitation, it is not considered that it detracts significantly from the prototype's capability to provide the focal point for constructive criticism. Naturally, this limitation of Prototype 1 will rectified for the final system.

### B.5.2 Information Stored in Cases

As the purpose of Prototype 1 is to demonstrate functionality, the information stored in the cases is only sufficient to permit this to be achieved. That is, to show that information can be entered into cases and then queried at a later date.

### B.5.3 Weightings Applied to Constraints

For the operation of Prototype 1 numerical weightings for constraints have been chosen at random i.e., the size and range of constraint weightings is not supported by reasoning and there is no rigorous process to support selection. It is acknowledged that the final system, if it uses numerical weightings, must have weights derived with an appropriate level of justification.

## B.6. Prototype 1 Development Tools

The following sections outline the software and hardware tools used to code the first ICES prototype.

### B.6.1. Hardware

Prototype 1 was developed using 486 and Pentium PCs.

### B.6.2. Software

Prototype 1 was developed using Borland Turbo C++ for Windows, version 3.1.

### B.6.3. Comment

For future development of the prototype it will be necessary to transfer the code to the UNIX workstation environment. This is necessary for two reasons:
1, To avoid run time memory problems that where encountered when developing Prototype 1 on a PC in a Windows environment.
2, The specification for the research project requires that the final system runs in a UNIX environment.

It was decided to develop Prototype 1 in a PC environment for the simple reason that the author had readier access to a PC than a workstation. The plan being to transfer the code from the PC across to the workstation on completion of Prototype 1. However, on transferring the code to a workstation it was found that the Borland C++ compiler was not compatible with the UNIX C++ compiler. The result of this incompatibility between the compilers is the requirement to debug the Borland compiled code in order to make it compatible with the UNIX compiler. It should be noted that the code itself is not incorrect but rather that some compilers are more 'fussy' than others. The debugging of the Borland compiled code on the UNIX workstation will probably only require the relocation of certain pieces of code within the program. However, as the prototype code is over 100 pages this could be a time consuming business and it was considered that this would be best performed after Prototype 1 has been critiqued.

## B.7. Step-through Examples of ICES Prototype 1, Knowledge-Based System (KBS) and Case Base Operation

### B.7.1. Prototype 1 - Start Up

Prototype 1 may be started by double clicking on the Proto1.exe icon in the Windows File Manager.

### B.7.2. Example 1 - KBS Operation

**1,**

MAIN MENU
1. KBS Operation
2. Case Base Operation
3. Quit

Please Make Selection:

*Enter 1 at the prompt*

**2,**

Enter a number corresponding to one of the following and press RETURN.

Terms relating to what the structure under consideration provides for the aircraft.

1. agility
2. subsonic
3. supersonic

Terms relating to the broader company view of design

4. cost
5. core-competences

*Enter 1 at the prompt*

**3,**

To achieve 'agility' there is a requirement for
a good instantaneous turn rate and a good sustained
turn rate. Both require a low wing loading

wing loading = weight/ wing area
wing loading is minimised by REDUCING WEIGHT.

The default weight constraint is 10:

Do you wish to change the constraint value? (y/n)

*Enter n at the prompt*
*Continue to answer the questions following this one with either y or n*

**4,**

another term? (y/n):

*Enter y at the prompt*

**5,**

Enter a number corresponding to one of the following and press RETURN.

Terms relating to what the structure under consideration provides for the aircraft.

1. agility
2. subsonic
3. supersonic

Terms relating to the broader company view of design

4. cost
5. core-competences

*Enter 4 at the prompt*

**6,**

Costs are directly affected by the manufacturing process used
and this in turn is influenced by the numbers produced,
i.e. One-off, Batch or Mass Production

The default production constraint is batch production
The default batch weighting is: 6
Do you wish to maintain batch production as the default? (y/n)

*Enter y at the prompt*

Do you wish to alter the batch production weighting? (y/n)

*Enter n at the prompt*

*Continue to answer the questions following this one with either y or n and enter new constraint weighting values as instructed.*

**7,**

another term? (y/n):

*Enter y at the prompt*

**8,**

Enter a number corresponding to one of the following and press RETURN.

Terms relating to what the structure under consideration provides for the aircraft.

1. agility
2. subsonic
3. supersonic

Terms relating to the broader company view of design

4. cost
5. core-competences

*Enter 5 at the prompt*

**9,**

In the development of a new design the degree of
R&D input will reflect one of the following requirements:

1. The new design is to be developed relying solely
on existing core competences
R&D Level 1: R&D input will be minimised

2. The development of the new design will aim to
expand existing competencies.
R&D Level 2: R&D input is allowable only with existing structures

3. The development of the new design will be looking at
developing a totally new structure and thus gaining new
core competences

The default R&D level is R&D Level 2
The default constraint for R&D Level2 is: 5
Do you wish to maintain R&D Level 2 as the default level? (y/n)

*Enter y at the prompt*

Do you wish to alter the R&D Level 2 weighting? (y/n)

*Enter n at the prompt*

**10,**

another term? (y/n):

*Enter n at the prompt*

**11,**

KBS QUESTIONS

press RETURN for more

Can the structure or parts of the structure
be represented as a single homogenous structure? (y/n)

*Enter y at the prompt*

Does the structure have an enclosed load bearing
internal section? (y/n)

*Enter y at the prompt*

Will the structure have to support loads such as
torsion and bending that lie outside of the structural
load path of the spars and thus require ribs or comparable
stiffening medium? (y/n)

*Enter y at the prompt*

What is the maximum section wall thickness, in mm?

*Enter 6 at the prompt*

What is the minimum wall thickness, in mm?

*Enter 2 at the prompt*

What is the depth of the section, in mm?

*Enter 30 at the prompt*

Is it a prime requirement of the structure to be able to
dissipate heat? (y/n)

*Enter n at the prompt*

**12,**

KBS MENU
1. KBS Output
2. Show Current Constraints
3. Revise Current Constraints
4. Repeat KBS Questions
5. Show Default Constraints
6. Return to Main Menu

*Enter 1 at the prompt*

**13,**

From the constraint values and the answers to questions no structure could be recommended that both satisfied manufacturing requirements and structural requirements.
However, would you like to look at the justification for the preferred structure from a manufacturing point of view and the preferred structure from a structural point of view?

*Enter y at the prompt*

**14,**

From a manufacturing point of view SPF/DB is the preferred process
Do you wish to see the justification for this selection?

*Enter y at the prompt*

*The system will now cycle through the manufacturing justification for SPF/DB*

**15,**

From a structural point of view Cellular Core is the preferred structure
Do you wish to see the justification for this selection?

*Enter y at the prompt*

*The system will now cycle through the structural justification for Cellular Core.*

**16,**

The case which corresponds most closely to your requirements is:

Core type: spaced-x-core

Maximum positive shear load: 59.794kN
Maximum negative shear load: -59.901kN
Maximum positive bending load: 37.999kNm
Maximum negative bending load: -37.942kNm
Maximum positive torque load: 8.01kNm
Maximum negative torque load: -8.095kNm
Tensile yield stress: 775Mpa
Ultimate tensile strength: 845Mpa
Young's Modulus: 107350Mpa
Density: 4.45Mg/$m^3$

press RETURN to continue

**17,**

KBS MENU
1. KBS Output
2. Show Current Constraints
3. Revise Current Constraints
4. Repeat KBS Questions
5. Show Default Constraints
6. Return to Main Menu

*The user may select any of the above operations as desired; alter the constraints acting on the process and then select option 1.*

*On completion of interaction with rule base select option 6.*

## 7.3. Example 2 - Case Base Operation

**1,**

MAIN MENU
1. KBS Operation
2. Case Base Operation
3. Quit

Please Make Selection:

*Enter 2 at the prompt*

**2,**

CASE BASE MENU
1. Enter New Case
2. Query Case Base
3. Return to Main Menu

Please Make Selection:
*Enter 2 at prompt*

**3,**

CASE BASE QUERY

Enter case details as prompted.

If prompted data not known enter ZERO.

Enter maximum positive shear load, in kN:

*Enter a value for shear load and press RETURN*

*Continue entering values as prompted by the system*

**4,**

The system will present the best matching stored case, e.g.

Core type: spaced-x-core
Maximum positive shear load: 59.794kN
Maximum negative shear load: -59.901kN
Maximum positive bending load: 37.999kNm
Maximum negative bending load: -37.942kNm
Maximum positive torque load: 8.01kNm
Maximum negative torque load: -8.095kNm
Tensile yield stress: 775Mpa
Ultimate tensile strength: 845Mpa
Young's Modulus: 107350Mpa
Density: 4.45Mg/$m^3$

**5,**

CASE BASE MENU
1. Enter New Case
2. Query Case Base
3. Return to Main Menu

Please make selection:

*The user may select option 1 in order to input a new case. It should be noted however that the append option has been omitted from prototype code. As such, any new case will over write existing cases.*

*Option 2 should be selected if the user wishes to further query the case base.*

# Appendix C

# ICES Software Implementation

# Appendix C - ICES Software Implementation

This appendix discusses the underlying code that supports the operation of the ICES software prototype. The appendix is divided into five sections. The first section provides a description of the underlying structure of the software supporting the second ICES prototype. The second section discusses the code that supports the ICES prototype 'front-end'. This is followed by two further sections which discuss the operation of the code that supports the ICES KBS and case base capability. The final section in this appendix discusses how the code and supporting 'visual' components of the ICES prototype could be expanded such that the system may support additional functionality beyond that which has been presented in this thesis.

## C.1 The Structure of the ICES Software

As indicated in Chapter 8, C++ Builder supports the development of C++ applications in a 'visual' environment. However, as indicated in section D.1.1.4 of appendix D, C++ Builder poses limitations with respect to building applications. The limitation which had the most impact on the structure of the ICES prototype was VCL's inability to support multiple inheritance like C++ does. This problem with inheritance proved to have more impact on the prototype development than first envisaged. While VCL's failure to support multiple inheritance was irritating it was not considered to be an irredeemable problem; it was envisaged that single inheritance would suffice. However, during the initial stages of coding the ICES prototype it became apparent that VCL components in reality do not support any form of inheritance. This is because VCL components are themselves derived through the object oriented methodology and the commonly used VCL components (e.g. dialog boxes) already support one line of inheritance i.e., this line of inheritance provides for their very existence. Clearly, when the application developer defines his or her own classes with data which it is desired to pass via inheritance to a VCL component then this constitutes multiple inheritance; the result being that the compiler will indicate an error.

This limitation in terms of inheritance did not mean that the ICES prototype application could not be developed within C++ Builder. However, it did determine that the software could not be coded using truly object oriented programming techniques. What this meant in terms of coding the ICES prototype was that it was not possible to pass information from C++ classes or other VCL components into VCL components, in particular dialog boxes, through normal inheritance methods. In order to overcome this limitation it was necessary for VCL components to pass numerical 'flags' to global variables. When another VCL component recognised that an appropriate flag was present it then could initiate some action. The global variables were all contained within a range of *structures* located in a header file called *buffer.h*. A typical *structure* residing within the *buffer.h* file is indicated below.

```
typedef struct
{
int startup_ir[50];
int materials_ir[50];
int labour_ir[50];
int processing_time_ir[50];
```

```
int part_reduction_ir[50];
int inspection_ir[50];
int assembly_ir[50];
}COST_STRUCT_IR_SELECTIONS;
```

A real-world application such as the ICES prototype usually has many source files containing the program's code. A global variable declared in one source file is global to that file but is not visible to any other source files used within the application. Clearly, in order for the various source files within the ICES prototype to be able to 'see' the values of global variables initiated by other source files it is necessary to have a method by which global variables generated by one source file are accessible to all others who need to use these variables. To make a global variable defined in one source file visible to all other source files that need to use that variable is a two-step process. Firstly, the variable is declared within one source file as global in the normal manner. Secondly, in any other source file that needs access to that global variable, the variable is declared again but this time with the *extern* keyword:

```
extern int count;
```

The extern keyword tells the compiler that the program is going to use a variable in one particular source file but the compiler will find the variable in question declared in another source file.

As indicated above, all the global variables used within the ICES prototype were placed within *structures* and therein one particular header file i.e., *buffer.h*. The reason for placing global variables in *structures* was that it facilitated the partitioning of the variables. Clearly, global variables act on different parts of the program and by keeping related variables together this facilitates the ease of programming and subsequent documentation and understanding. Global variables residing in structures still employ the *extern* keyword. However, the syntax is slightly differently than for a stand-alone variable:

```
extern COST_STRUCT_IR_SELECTIONS cost_ir_select;
cost_ir_select.part_reduction_ir[0];
```

In this instance, COST_STRUCT_IR_SELECTIONS is the name of the *structure* and cost_ir_select points to this *structure.* The segment of code cost_ir_select.part_reduction_ir[0] identifies the array within the *structure* where variables may be assigned. For the same reason that similar global variables where placed in the same *structure*, the placing of all global variables in the same header file again facilitates the ease of programming and understanding. Clearly, if in the future it is desired to further expand the ICES prototype the developer does not want to be searching through all the program source files in order to locate all the global variables; far better that they are placed in one location.

It should be pointed out that even if the ICES prototype were coded using the full range of object oriented programming techniques available, it is still highly likely that there would be a requirement for global variables. However, it is likely that they

would not be used to the same extent as they have been in this particular software implementation of the ICES methodology.

## C.2 ICES Software - 'Front-end'

This section discusses the code that supports the operation of the ICES prototype's 'Front-end'.

### C.2.1. File menu options - Exit

The Exit menu option permits in the File menu permits the user to close down the ICES prototype. The code that facilities this exit capability is as follows:

```
void __fastcall Tices_mainform::ExitClick(TObject *Sender)
{
        Close();
}
```

When the user clicks on the Exit sub-menu option the above function is called and the Close() command is initiated. It should be pointed out that the Close() command ends the operation of the currently active dialog box. C++ Builder does not discriminate between dialog boxes and windows. The software package sees them as one of the same. As C++ Builder is constructed using the object oriented methodology where VCL components are created in a hierarchical manner through the application of inheritance, the Close() command has the effect of moving the application to next higher level in the hierarchy. In the case of the operation of the Close() command initiated by clicking on the Exit sub-menu there is no higher level in the hearty for the application to move to, and as such the application shuts down.

### C.2.2. Component menu options - Enter Component

In order to facilitate the understanding of the operation of the Enter Component sub-menu and the Structure Identification dialog box the code relating to these operations described above is now given:

```
void __fastcall Tices_mainform::EnterComponent1Click(TObject *Sender)
{
        struct_ident->ShowModal();
}
```

When the user clicks on Enter Component sub-menu the above function is called. The term *struct_ident* is the name by which the application recognises the Structure Identification dialog box. This *struct_ident* term points to the command *ShowModal()*. The command *ShowModal()* is the command that initiates the display of a dialog box in the application i.e., the user can see it and now interact with it. In the context of the above code example, the *struct_ident* term points to the *ShowModal()* command and if *ShowModal()* 'sees' the dialog box that relates to the *struct_ident* term i.e., the Structure Identification dialog box, this is then displayed.

When the user enters the component it is desired to design in the Structure Identification dialog box this has the effect of enabling menu options residing beneath the KBS main menu option. The code that supports this capability is given below:

```
void __fastcall Tstruct_ident::BitBtn1Click(TObject *Sender)
{
        AnsiString s;
        int Size = Edit1->GetTextLen();
        char *name = new char[++Size];
        Edit1->GetTextBuf(name,Size);
        s=name;
        delete name;

        if(s == "foreplane")
        {
        aero_struct.foreplane[0] =1;
        }
}
```

The above code relates directly to the Structure Identification dialog box as indicated in Figure 9.3. When the user enters the component it is desired to design in the edit box of the Structure Identification dialog box and clicks on the OK button or presses return the function above is invoked. The code *AnsiString s* refers to a string object *s* instantiated from the AnsiString class. The function then declares the integer variable *Size* and through the function call *GetTextLen()* gets the length of the string entered in the edit box *Edit1* and assigns this value to *Size*. The code *char *name = new char[++Size]* allocates sufficient space for the string in memory; **name* is a pointer to the memory buffer. The line of code *Edit1->GetTextBuf(name,Size)* places the string in the edit box into the memory buffer. The string stored in the memory buffer *name* is assigned to the string object *s*. Memory allocated to the buffer is then freed with the command *delete name*. The if statement *if(s == "foreplane")* compares the string object *s* to the string *foreplane*. If *s* equals the *foreplane* string then the value 1 is assigned to a global variable residing in a structure array in the header file *buffer.h*. As will be illustrated below this variable can then seen by the main menu components and thus acted upon i.e., facilitating the enabling of the KBS menu option in particular.

As mentioned in several places in this thesis it was decided to select the foreplane as the aircraft structure to drive through the ICES software prototype in order to validate the methodology described in Chapter 7. Hence, the string *foreplane* was provided in the above coded example. With relatively minor changes to the ICES prototype code the system could readily facilitate the introduction of other structure types e.g., wing. Section C.5.1 describes how the code presented so far could be expanded to support other aircraft structures.

### C.2.3. KBS

At start-up all the pull-down menu options in the KBS menu are disabled by default. Whether a menu is enabled or disabled at run-time is determined by the application

developer at build-time. The Object Inspector as discussed in section D.1.1.3 of appendix D enables a VCL component's properties to be defined by the application developer.

As indicated in section C.2.2, the entering of an appropriate component i.e., a descriptive string, in the Structure Identification dialog box causes the value 1 to be assigned to a global variable residing in a structure array in the header file *buffer.h*. As can be seen from the following code example and explanation below that, the ability of the KBS main menu component to see this global variable allows the application to enable the KBS sub-menu options.

```
void __fastcall Tices_mainform::kbsClick(TObject *Sender)
{
        if(aero_struct.foreplane[0]==1)
        {
        DesignDriverSelection1->Enabled = true;
        Question1->Enabled = true;
        MatrixOutput1->Enabled = true;
        best_structure-> Enabled = true;
        }
}
```

As outlined in the discussion concerning the Component main menu option i.e., in the context of the operation of the Structure Identification dialog box discussed in section C.2.2, if the string object *s* equals the *foreplane* string the value 1 is assigned to *aero_struct.foreplane[0]*. This being a global variable residing in a structure array in the header file *buffer.h*. This global variable may now be 'seen' by the main menu components. If the user now clicks on the KBS menu option the function indicated above is called. The function first tests to see if the value assigned to the variable in the structure array *aero_struct.foreplane[0]* equals 1 i.e., *if(aero_struct.foreplane[0]==1)*. If the value assigned to the variable in the structure array *aero_struct.foreplane[0]* does equal 1 it is then possible to access the *if* conditional statement. As indicated, all the KBS menu options are disabled on start-up; by accessing the *if* conditional statement it is now possible to enable these menu options. As an example, consider the code *DesignDriverSelection1->Enabled = true*. Here the term *DesignDriverSelection1* is the name by which the application recognises the Design Driver Selection menu option in the KBS menu. The term *DesignDriverSelection1* points to the *Enabled* command. This Boolean command can be either *true* or *false*. In this instance *Enabled* is assigned the value *true*. This then enables the Design Driver Selection menu option i.e., the user will see the menu option become non-grey and active and the operations that this menu command supports may now be accessed. It can be seen from the above code example that the operation described for enabling the Design Driver Selection sub-menu holds true for all the other menu options in KBS menu.

### C.2.3.1. KBS menu options - Design Driver Selection

As discussed in section 9.2.1.3.1 of Chapter 9, the Design Driver Selection is the first KBS menu option. It provides access to a further sub-menu, see Figure 9.4. This sub-

menu is disabled on application start-up but is enabled in conjunction with the other KBS menu options. As can be seen from section C.2.3, the code that enables these sub-menu options is identical in format to that which is used to enable the KBS menu options. The only difference being the menu options themselves.

```
void __fastcall Tices_mainform::DesignDriverSelection1Click(TObject *Sender)
{
        if(aero_struct.foreplane[0]==1)
        {
        Agility1->Enabled = true;
        Supersonic1->Enabled = true;
        Subsonic1->Enabled = true;
        CoreCompetences1->Enabled = true;
        Cost1->Enabled = true;
        }
}
```

The selection of the Design Driver Selection Sub-menu options i.e., Agility, Supersonic, Subsonic, Core-Competences and Cost, activate a series of dialog boxes. The code that initiates these dialog boxes from the Design Driver Selection sub-menu is given below:

*Agility*

```
void __fastcall Tices_mainform::Agility1Click(TObject *Sender)
{
        agility->ShowModal();
}
```

*Supersonic*

```
void __fastcall Tices_mainform::Supersonic1Click(TObject *Sender)
{
        supersonic->ShowModal();
}
```

*Subsonic*

```
void __fastcall Tices_mainform::Subsonic1Click(TObject *Sender)
{
        subsonic->ShowModal();
}
```

*Core-Competences*

```
void __fastcall Tices_mainform::CoreCompetences1Click(TObject *Sender)
{
        r_d_input->ShowModal();
}
```

*Cost*

```
void __fastcall Tices_mainform::Cost1Click(TObject *Sender)
{
```

```
        cost_input->ShowModal();
}
```

The above five functions relate to each one of the Design Driver Selection sub-menu options. When a menu option is selected the appropriate function is called. It can be seen that the underlying operation of these functions is identical to that employed to display the Structure Identification dialog box which is discussed in section C.2.2. Rather than repeat the explanation of how this code operates the reader is referred to this section.

### C.2.3.2. KBS menu options - Questions

As discussed in section 9.2.1.3.2 of Chapter 9, the Questions menu option in the KBS menu provides access to the Questions dialog box. When the user clicks on this menu option the Questions dialog box is presented. This dialog box asks the user a series of questions that relate to the component entered in the Structure Identification dialog box. The remainder of this section now discusses the code that provides the underlying functionality of the Questions dialog box.

Clearly, the user's answers to the questions in the Question dialog box must be accessed by other parts of the KBS. This is achieved by assigning the answers provided by the user to global variables contained in a *structure* located in the header file *buffer.h*. The code that provides the Question dialog box functionality is now given below.

```
void __fastcall Tquestions::okay_buttonClick(TObject *Sender)
{
int len;
float qa, qb, qc;
char buffer1[30];
char buffer2[30];
char buffer3[30];
char buffer7[30];
qa=get_number(Edit4);
qb=get_number(Edit5);
qc=get_number(Edit6);
kbs_questions.max_wall_thickness[0] = qa;
kbs_questions.question_answer5[0] = qb;
kbs_questions.question_answer6[0] = qc;
len = Edit1->GetTextBuf(buffer1,30);
len = Edit2->GetTextBuf(buffer2,30);
len = Edit3->GetTextBuf(buffer3,30);
len = Edit7->GetTextBuf(buffer7,30);

        if(buffer1[0] == 'y')
        {
        kbs_questions.homogenous_structure[0] = 1;
        }
        else
```

```
{
kbs_questions.homogenous_structure[0] = 0;
}

if(buffer2[0] == 'y')
{
kbs_questions.internal_section[0] = 1;
}
else
{
kbs_questions.internal_section[0] = 0;
}

if(buffer3[0] == 'y')
{
kbs_question.load_path[0] = 1;
}
else
{
kbs_questions.load_path[0] = 0;
}

if(buffer7[0] == 'y')
{
        kbs_questions.question_answer7[0] = 1;
        }
        else
        {
        kbs_questions.question_answer7[0] = 0;
        }
}
```

As indicated above, when the user clicks on the Questions menu option the Questions dialog box is presented. The user should answer the questions presented in the dialog box in the manner requested by each particular question. This will be either *y* or *n* for yes or no, or a numerical value. Once all the questions have been answered the user should click on the OK button. This button initiates the function indicated above. It can be seen that the function declares one integer variable *len*, three float variables *qa*, *qb* and *qc* and also four character arrays *buffer1[30]*, *buffer2[30]*, *buffer3[30]* and *buffer7[30]*.

Taking the function in sequence. The float variables are provided to handle the answers to questions that are expressed in millimetres. In addition, it can be seen that there are three calls to the function *get_number*, the output of which is assigned to each of the three float variables e.g., *qa=get_number(Edit4)*. The purpose of the *get_number* function is to convert those numerical entries in the Questions dialog box edit boxes to floating point numbers. This operation may appear rather strange to the reader. However, things should be clearer if it is appreciated that any character be it text or a number is seen by the system as just a string. The *get_number* function

provides access to a further function *atof* which converts strings to floating point numbers.

Once the *get_number* function has assigned three floating point numbers to the three float variables i.e., *qa*, *qb*, and *qc*, these values are then assigned to the global variable arrays residing in the *structure* QUESTION_RESPONSE which is defined in the header file *buffer.h*. Thus, the variables now become accessible to all other parts of the KBS that need to know the response to the numerically based questions asked in the Questions dialog box.

The line of code *len = Edit1->GetTextBuf(buffer1,30)* is repeated four times as it corresponds to the four edit boxes in the Questions dialog box which require a character to be entered i.e., y or n. The code determines the length of the string in the edit box and proceeds to put the string into the character array *buffer1*. The amount of space available in the character array is declared as being 30.

The remaining code in the above function compares the characters residing in *buffer1*, *buffer2, buffer3* and *buffer7* with the character 'y' in four separate conditional *if* and *else* statements. If the characters residing in the character arrays *buffer1*, *buffer2*, *buffer3* and *buffer7* do indeed equal 'y' then it is possible then to enter the conditional statements and the value 1 is assigned to the appropriate global variable array declared in the *structure* QUESTION_RESPONSE which itself is declared in *buffer.h*. However, if the characters residing in *buffer1*, *buffer2, buffer3* and *buffer7* do not equal 'y' then the program is directed to the *else* part of the *if else* statement and a zero is assigned to the global variable array. Clearly, by assigning 1s and zeros to the global variable arrays within the *structure* QUESTION_RESPONSE all other components within the KBS that need to know the response to yes/no questions in the Questions dialog box can do so.

The *get_number* function shown below, and initially mentioned above, provides the capability to convert strings into floating point numbers. Taking a line of code at a time. The function declares a character array called *buffer* with space for thirty characters. This is followed by the declaration of a float variable *f*. The code *len = Edit1->GetTextBuf(buffer,30)* as indicated above, determines the length of the string in the edit box and places the string into the character array *buffer*. The function call *atof(buffer)* converts the string residing in *buffer* to a floating point number and assigns it to the float variable *f*. The value of float *f* is then returned to the function that originally called the *get_number* function by *return f*.

```
float __fastcall Tquestions::get_number(TEdit *edit)
{
        char buffer[30];
        float f;
        int len = edit->GetTextBuf(buffer,30);
        f = atof(buffer);
        return f;
}
```

As already mentioned, the *structure* QUESTION_RESPONSE shown below is declared in the header file *buffer.h*. As intermated at the beginning of this appendix, the variables assigned to this *structure* can be made global and thereby accessed by all other parts of the ICES prototype application that needs to use them.

```
typedef struct
{
int homogenous_structure[50];
int internal_section[50];
int load_path[50];
float max_wall_thickness[50];
float question_answer5[50];
float question_answer6[50];
int question_answer7[50];
}QUESTION_RESPONSE;
```

In the discussion relating to the operation of the Questions dialog box in section 9.2.1.3.2 of Chapter 9, the reader's attention was focused on the push-button controls that reside in the dialog box, in particular the Reset button. This button permits the user to cancel previous answers to the questions and start again if so desired. In fact, operating the Reset button places zeros in all the edit boxes residing in the Questions dialog box. This capability is achieved through the function shown below.

```
void __fastcall Tquestions::BitBtn2Click(TObject *Sender)
{
float quest1_reset,quest2_reset,quest3_reset,quest4_reset,quest5_reset;
float quest6_reset,quest7_reset;
quest1_reset = 0;
quest2_reset = 0;
quest3_reset = 0;
quest4_reset = 0;
quest5_reset = 0;
quest6_reset = 0;
quest7_reset = 0;
Edit1->Text = quest1_reset;
Edit2->Text = quest2_reset;
Edit3->Text = quest3_reset;
Edit4->Text = quest4_reset;
Edit5->Text = quest5_reset;
Edit6->Text = quest6_reset;
Edit7->Text = quest7_reset;
}
```

When the Reset button is operated the above simple function is called. The function first declares seven float variables and these are then assigned the value zero. The variables are then output to the edit boxes with seven versions of the code *Edit1->Text = quest1_reset.*

### C.2.3.3. KBS menu options - Matrix Output

The principle components of the code that supports underlying operation of the Design Driver Ranking Matrix Output dialog box is given below.

```
typedef struct
{
float MinWeightInformation1[50];
}STRUCT_MW_INFO;

typedef struct
{
float High_SS_Information1[50];
}STRUCT_HSS_INFO;
```

As examples, the two global variable arrays residing in the *structures* shown above, which themselves reside in the *buffer.h* header file, provide access to the mean average of the impaction scores relating to the minimum weight and high structural strength design drivers. These scores can be 'seen' by the Design Driver Ranking Matrix Output dialog box and can thus be entered into the Average edit boxes as described in section 9.2.1.3.3 of Chapter 9. Within the *buffer.h* header file there are similar *structures* to the ones shown above which correspond to the rest of the design drivers currently embraced by the Design Driver Ranking Matrix Output dialog box.

```
typedef struct
{
float SensitivityInformation1[50]; //minimum weight
float SensitivityInformation2[50]; //high structural strength
float SensitivityInformation3[50]; //low torque
float SensitivityInformation4[50]; //thin section
float SensitivityInformation5[50]; //one-off production
float SensitivityInformation6[50]; //batch production
float SensitivityInformation7[50]; //mass production
float SensitivityInformation8[50]; //start-up
float SensitivityInformation9[50]; //material
float SensitivityInformation10[50]; //labour
float SensitivityInformation11[50]; //part reduction
float SensitivityInformation12[50]; //processing time
float SensitivityInformation13[50]; //inspection requirements
float SensitivityInformation14[50]; //assembly
}STRUCT_SEN_INFO;
```

The global variable arrays residing in the *structure* STRUCT_SEN_INFO shown above, which itself is declared in the *buffer.h* header file, provide access to the design driver sensitivity scores. These scores can be 'seen' by the Design Driver Ranking Matrix Output dialog box and can thus be entered into the Sensitivity edit boxes as described in section 9.2.1.3.3. The abridged function shown below illustrates how the Minimum Weight design driver importance rating is calculated; the method shown is identical for all other design drivers.

```
void __fastcall Tmatrix::FormActivate(TObject *Sender)
{
float mw_sens; //sensitivity variable
float matmwa, mwavg, mwsum, mwadd_avg_sum; //minimum weight variables
float mwsens, mwavgsum, mwupdate, mwimp_rat;

//assign minimum weight value from buffer.h to variable
matmwa=mw_info.MinWeightInformation1[0];

//assign sensitivity value from buffer.h to variable
mw_sens=sense_info.SensitivityInformation1[0];

        //if sensitivity buffer not empty continue
        if(mw_sens != '\0')
        {
        //write contents of buffer to edit box
        Edit22->Text=mw_sens;
        //overwrite sensitivity buffer
        sense_info.SensitivityInformation1[0] = '\0';
        }

        //if minimum weight buffer not empty
        if(matmwa != '\0')
        {
        int mw_update, r, x;
        r = 1;

        //write contents of buffer to edit box
        Edit1->Text = matmwa;

        //get contents of update edit box
        mw_update = get_number(Edit15);
        //increment the update value by 1
        x = mw_update + r;
        Edit15->Text = x;

        mwavg = get_number(Edit1);
        mwsum = get_number(Edit8);
        //calculate the new mean average value of minimum weight value scores
        mwadd_avg_sum = mwavg + mwsum;
        Edit8->Text = mwadd_avg_sum;

        mwsens = get_number(Edit22);
        mwavgsum = get_number(Edit8);
        mwupdate = get_number(Edit15);

        //calculate the ddrt importance rating
        mwimp_rat = sqrt(((mwavgsum / mwupdate) * mwsens));
        Edit29->Text = mwimp_rat;
```

```
        //assign null value to buffer to ensure update value works correctly
        mw_info.MinWeightInformation1[0] = '\0';
        }
}
```

Taking the above function in sequence. The function first declares nine float variables. The function then takes the value residing in the *structure* STRUCT_MW_INFO residing in the *buffer.h* header file and assigns it to the float variable *matmwa* i.e., *matmwa=mw_info.MinWeightInformation1[0]*. In a similar manner, the function takes the value residing in the *structure* STRUCT_SEN_INFO which again resides in *buffer.h* and assigns it to the float variable *mw*_sens i.e., *mw_sens=sense_info.SensitivityInformation1[0]*.

The function leads onto test the value assigned to the variable *mw_sens* i.e., this variable corresponds to the sensitivity of the minimum weight design driver. The function uses the *if* conditional statement *if(mw_sens != '\0')* to test the value of *mw_sens*. If the value of *mw_sens* does not equal *null* i.e., implying there is a value residing in the array *SensitivityInformation1*, then the conditional statement can be entered. Assuming that the value of the float variable *mw_sens* does not equal *null*, *mw_sens* is written to the Sensitivity edit box corresponding to the minimum weight design driver. Having written the value of *mw_sens* to the Sensitivity edit box, a *null* value is now assigned to the array *SensitivityInformation1*. By assigning a *null* value to the *SensitivityInformation1* array the program is now denying access to the *if* conditional statement until the sensitivity variable array residing in structure STRUCT_MW_INFO is assigned a new sensitivity score from an external source i.e., the Sensitivity dialog box which is discussed in section 9.2.1.4.2 of Chapter 9.

The function now tests the value assigned to the float variable *matmwa* in a similar manner to the way the float variable *mw_sens* was tested above i.e., with an *if* conditional statement. Assuming that *matmwa* does equal *null* the *if* conditional statement *if(matmwa != '\0')* is entered. It can be seen that the function now declares three integer variables i.e., *mw_update*, *r* and *x*. The integer variable *r* is assigned the value 1. The code *Edit1->Text = matmwa* now writes the value held by the float variable *matmwa* to the Average edit box corresponding to the minimum weight design driver.

Utilising the *get_number* function, the value residing in the Updates edit box is obtained and assigned to the integer variable *mw_update* with the code *mw_update = get_number(Edit15)*. The *get_number* function is discussed in detail in section C.2.3.2. As can be seen from the above function, the value assigned to the *mw_update* integer variable is incremented by 1 and the resulting value is assigned to the integer variable *x* i.e., $x = mw\_update + r$. This new updated value assigned to the integer variable *x* is now written to the Update edit box corresponding to the minimum weight design driver with the code *Edit15->Text = x*.

The function now leads on to call the *get_number* function twice, obtaining the values residing in the Average and Sum edit boxes and assigning them to the *mwavg* and *mwsum* float variables respectively. The function now proceeds to add the values assigned to these two variables together and place the resulting solution back in the

Sum edit box i.e., *mwadd_avg_sum = mwavg + mwsum* and *Edit8->Text = mwadd_avg_sum.*

In order to calculate the minimum weight design driver importance weighting the program calls the *get_number* function three times, obtaining the values residing in the Sensitivity, Sum and Update edit boxes. These values are assigned to the float variables *mwsens*, *mwavgsum* and *mwupdate.* Using these values the *code mwimp_rat = sqrt(((mwavgsum / mwupdate) * mwsens))* calculates the minimum weight design driver importance weighting and assigns the result to the float variable *mwimp_rat.* The code *Edit29->Text = mwimp_rat* places the value assigned to the variable *mwimp_rat* in the Importance Rating edit box.

The function concludes with the code *mw_info.MinWeightInformation1[0] = '\0'.* This code assigns a *null* value to the array *MinWeightInformation1* which is declared in the *structure* STRUCT_MW_INFO which as indicated above resides in the header file *buffer.h.* The reason for assigning a *null* value to the array *MinWeightInformation1* is to ensure that the Update edit box can only be incremented when a new value is assigned to the *MinWeightInformation* array from an external source i.e., the Minimum Weight Default Update dialog box which is discussed in section 9.2.1.4.1 of Chapter 9.

Clearly, when the Design Driver Ranking Matrix Output dialog box is created as a result of the selection of the Matrix Output menu option from the KBS menu it is desirable that the numerical values that were present in the edit boxes on close down should be re-instigated on start-up. To provide this capability it is necessary to read and write the values residing in all the edit boxes in the Design Driver Ranking Matrix Output dialog box to and from file. The abridged function below shows the code necessary to write the contents of edit boxes to file.

```
void __fastcall Tmatrix::FormClose(TObject *Sender, TCloseAction &Action)
{
ofstream ofs("data1.txt", ios::binary);
ofs<<Edit1->Text<<' '
   <<Edit2->Text<<' '
   <<Edit3->Text<<' '
   <<Edit4->Text;
}
```

The code above declares *ofs* as an instantiated object of the class *ofstream* which is declared in the header file *iostream.h.* The object *ofs* opens the file *data1.txt* in binary mode and writes the contents of the edit boxes (*Edit1* to *Edit4*) to it. It is important to note that this function is only called when the user closes the Design Driver Ranking Matrix Output dialog box. Clearly, the full function that writes the contents of all the edit boxes in Design Driver Ranking Matrix Output dialog box is considerable larger than that shown above, handling the contents of all seventy edit boxes. The abridged function below shows the code necessary to read back the contents of the file data1.txt and assign it to the appropriate edit boxes in the Design Driver Ranking Matrix Output dialog box.

```
void __fastcall Tmatrix::FormCreate(TObject *Sender)
{
ifstream ifs("data1.txt", ios::binary);
string word;
ifs>>word;
Edit1->Text = word.c_str();
ifs>>word;
Edit2->Text = word.c_str();
ifs>>word;
Edit3->Text = word.c_str();
ifs>>word;
Edit4->Text = word.c_str();
}
```

The function above declares *ifs* as an instantiated object of the class *ifstream* which is declared in the header file *iostream.h*. The object *ifs* opens the file *data1.txt* in binary mode and reads the contents as strings to the edit boxes *Edit1* to *Edit4*. That is, the function declares *word* as an instantiated object of the class *string* which is declared in the header file *cstring.h*. The purpose of *word* is to enable the *ifs* object to stream the contents of the file *data1.txt* as strings. The above function is the converse of the function that writes the values from the edit boxes to file in that it is called on creation of the Design Driver Ranking Matrix Output dialog box.

A further aspect of the Design Driver Ranking Matrix dialog box that should be noted is that the design driver importance ratings are also written to another separate file in addition to the file *data1.txt* indicated above i.e., *data17.txt*. The reason for this is that these values need to be used to assist with the determination of the best structure. In the context of the structure of the ICES prototype the easiest way to transport these values to their destination where they are to be used is through standard file handling methods. Section 9.2.2.4 of Chapter 9, which discusses in detail how the best structure is derived, sees the file handling necessary to facilitate the use of these stored design driver importance ratings.

### C.2.4. Options

The two sub-sections in this section discuss the two pull-down menu options residing in the Options menu of the ICES software prototype.

#### C.2.4.1. Options menu options - Design Driver Matrix Update

As discussed in section 9.2.1.4.1 of Chapter 9, the Minimum Weight Default Update dialog box has four control push buttons. It is considered that the operation of the OK, Cancel and Help push buttons is self explanatory. However, the Update push button requires explanation. The purpose of the Update push button is to initiate the calculation of the mean average of the design driver impaction scores and then assign this mean average score to the appropriate structure array residing in the header file *buffer.h*. Thus, making the value of this score globally accessible to all those components of the ICES prototype which need to see it, specifically in this case, the

Design Driver Ranking Matrix Output dialog box. An example of the code that supports this capability is given below.

```
void __fastcall Tmin_weight_update::BitBtn2Click(TObject *Sender)
{
float mwa, mwb, mwc, mwd, mwe, mwf, mwg, mwh, mwi, mwj;
float mwk, mwl, mwm, mw_total;

mwa = get_number(Edit1);
mwb = get_number(Edit2);
mwc = get_number(Edit3);
mwd = get_number(Edit4);
mwe = get_number(Edit5);
mwf = get_number(Edit6);
mwg = get_number(Edit7);
mwh = get_number(Edit8);
mwi = get_number(Edit9);
mwj = get_number(Edit10);
mwk = get_number(Edit11);
mwl = get_number(Edit12);
mwm = get_number(Edit13);

mw_total = (mwa+mwb+mwc+mwd+ mwe+mwf+mwg+mwh+mwi+mwj+
            mwk+mwl+mwm) / 13;
mw_info.MinWeightInformation1[0] = mw_total;
}
```

The above code supports the operation of the Minimum Weight Default Update dialog box. However, this code is identical, apart from the float variable names, to that used with all other design driver Default Update dialog boxes. When the Update push button within the Minimum Weight Default Update dialog box is operated the above function is called. The function begins by declaring fourteen float variables. The function proceeds by then making thirteen calls to the function *get_number*. As indicated in section C.2.3.2, this function provides the capability to convert strings into floating point numbers. The float values returned by the calls to the *get_number* function are assigned to thirteen of the previously declared float variables. The values assigned to these float variables are then added together and divided by 13 i.e., this giving a mean average value. This mean average value is then assigned to the float variable *mw_total*. The value assigned to the variable *mw_total* is then itself assigned to the array *MinWeightInformation1* with the code *mw_info.MinWeightInformation1[0] = mw_total*. This array is declared in the *structure* STRUCT_MW_INFO which itself is declared in the header file *buffer.h*. As indicated above, this makes the value placed in the array *MinWeightInformation1* globally available to all components of the ICES prototype that need to use it.

It should be possible at this juncture for the reader to appreciate how values are passed from the Default Update dialog boxes to be used within the Design Driver Ranking Matrix Output dialog box. It is worth appreciating that this is the primary

method used in the ICES prototype for passing information from one VCL component to another. In majority of cases in the ICES prototype VCL components are represented by dialog boxes.

It will be necessary to be able to read and write the values residing in the edit boxes of all the Default and Default Update dialog boxes to and from file; specifically when dialog boxes are created and closed down. This operation is described in section C.2.3.3 which discusses the code supporting the Design Driver Ranking Matrix Output dialog box. It considered unnecessary here to repeat the discussion relating to reading and writing to and from edit boxes to file, instead the reader is directed to this section.

### C.2.4.2. Options menu options - Design Driver Sensitivities Update

As discussed in section 9.2.1.4.2 of Chapter 9, the Sensitivity dialog box has four push buttons i.e., OK, Cancel, Help, and an Update push button. It is considered that the operation of the OK, Cancel and Help push buttons is self explanatory. However, the Update push button requires explanation. The purpose of this control is to assign the numerical values residing in the edit boxes in the Sensitivity dialog box to the appropriate *structure* arrays which are declared in the header file *buffer.h*. Thus, making these sensitivity values globally accessible to all those components of the ICES prototype which need to access them, specifically the Design Driver Ranking Matrix Output dialog box as discussed in section 9.2.1.3.3 of Chapter 9. The code underlying the Sensitivity dialog box is given below.

```
void __fastcall Tsensitivity::BitBtn2Click(TObject *Sender)
{
float sena, senb, senc, send, sene, senf, seng, senh, seni, senj;
float senk, senl, senm, senn;

sena = get_number(Edit1);
senb = get_number(Edit2);
senc = get_number(Edit3);
send = get_number(Edit4);
sene = get_number(Edit5);
senf = get_number(Edit6);
seng = get_number(Edit7);
senh = get_number(Edit8);
seni = get_number(Edit9);
senj = get_number(Edit10);
senk = get_number(Edit11);
senl = get_number(Edit12);
senm = get_number(Edit13);
senn = get_number(Edit14);

sense_info.SensitivityInformation1[0] = sena;
sense_info.SensitivityInformation2[0] = senb;
sense_info.SensitivityInformation3[0] = senc;
sense_info.SensitivityInformation4[0] = send;
```

```
sense_info.SensitivityInformation5[0] = sene;
sense_info.SensitivityInformation6[0] = senf;
sense_info.SensitivityInformation7[0] = seng;
sense_info.SensitivityInformation8[0] = senh;
sense_info.SensitivityInformation9[0] = seni;
sense_info.SensitivityInformation10[0] = senj;
sense_info.SensitivityInformation11[0] = senk;
sense_info.SensitivityInformation12[0] = senl;
sense_info.SensitivityInformation13[0] = senm;
sense_info.SensitivityInformation14[0] = senn;
}
```

The above function is called when the Update push button in the Sensitivity dialog box is operated. The function begins by declaring fourteen float variables. This is followed by the function making fourteen calls to the function *get_number*. As indicated in section C.2.3.2 this function provides the capability to convert strings into floating point numbers. The float values returned by the calls to the *get_number* function are assigned to the fourteen float variables. The values assigned to these float variables are then themselves assigned to fourteen arrays which are declared in the *structure* STRUCT_SEN_INFO which resides in the header file *buffer.h*. The *structure* STRUCT_SEN_INFO is listed in section C.2.3.3. The values placed in *structure* arrays residing in the *buffer.h* header file may be made globally available to all components within the ICES prototype that need to use them. In this instance, the Design Driver Ranking Matrix Output dialog box is the principle interested party.

## C.3. KBS Operation

This section focuses on the code that facilitates the operation of the ICES KBS.

### C.3.1. Meta Rule Base

The design drivers introduced into the design process by clicking on the check boxes in the Agility dialog box must be made accessible to those down-stream components of the KBS. This is achieved through the following code.

*Minimum Weight*

```
void __fastcall Tagility::min_weight_cbClick(TObject *Sender)
{
int r;
r = 1;
        if(min_weight_cb->State == cbChecked)
        {
        agility_ir_select.agility_minimum_weight_ir[0] = r;
        }

        if(min_weight_cb->State == cbUnchecked)
        {
        agility_ir_select.agility_minimum_weight_ir[0] = '\0';
        }
```

```
}
```

*High Structural Strength*

```
void __fastcall Tagility::high_struct_strength_cbClick(TObject *Sender)
{
int b;
b = 2;
        if(high_structural_strength_cb->State == cbChecked)
        {
        agility_ir_select.agility_high_structural_strength_ir[0] = b;
        }
        if(high_structural_strength_cb->State == cbUnchecked)
        {
        agility_ir_select.agility_high_structural_strength_ir[0] = '\0';
        }
}
```

*Low Torque*

```
void __fastcall Tagility::low_torque_cbClick(TObject *Sender)
{
int c;
b = 3;
        if(low_torque_cb->State == cbChecked)
        {
        agility_ir_select.agility_low_torque_ir[0] = c;
        }

        if(low_torque_cb->State == cbUnchecked)
        {
        agility_ir_select.agility_low_torque_ir[0] = '\0';
        }
}
```

*Thin Section*

```
void __fastcall Tagility::thin_section_cbClick(TObject *Sender)
{
int d;
d = 3;
        if(thin_section_cb->State == cbChecked)
        {
        agility_ir_select.agility_thin_section_ir[0] = d;
        }

        if(thin_section_cb->State == cbUnchecked)
        {
        agility_ir_select.agility_thin_section_ir[0] = '\0';
        }
}
```

When a check box is clicked in the Agility dialog box, one of the above four functions is called. It is important to note that the default setting for all the check boxes is un-checked i.e., this default setting is set in the Object Inspector during code development. Taking the function relating to the minimum weight design driver as an example, and the code in order of declaration. Assuming the check box corresponding to the minimum weight design driver is clicked on, the function *min_weight_cbClick* is called. The function first declares the integer variable *r* and assigns the value 1 to it. This is followed by an *if* conditional statement test i.e., *if(min_weight_cb->State == cbChecked)*. The code *min_weight_cb* is the name by which the application recognises the minimum weight check box. *State* is a C++ Builder command for 'seeing' the condition of a check box. The *cbChecked* statement declares the state of the check box as being checked. If the *State* command equals the *cbChecked* statement i.e., the check box is indeed checked, the *if* conditional statement can then be entered. Having entered the *if* conditional statement the value 1 which is assigned to the variable *r* is itself assigned to the global array *agility_ir_select.agility_minimum_weight_ir[0]*. This array is declared in the header file *buffer.h*. This facilitates the variable value being visible to all other components in the KBS that need to use it.

It can be seen that there is a second *if* conditional statement in the function that relates to the minimum weight design driver. This second conditional statement declares the state of the check box as being un-checked. Clearly, if the minimum weight design driver check box is click on a second time the check box state will become unchecked. However, the function will still be called but this time this second *if* conditional statement will be entered and not the first. Here a *null* value is assigned to the *agility_minimum_weight_ir array*. This ability for the value assigned to the *agility_minimum_weight_ir* array to be changed depending on the user input permits other components in the KBS to determine whether a design driver declared in the Agility dialog box is active or not. It should be noted that if the check box corresponding to the minimum weight design driver is not clicked on at all during an application run the *agility_minimum_weight_ir* array will retain a default *null* value. This feature holds true for all design drivers declared in the Agility dialog box.

As with the Agility dialog box, the design drivers introduced into the design process by clicking on the check boxes in the Supersonic and Subsonic dialog boxes must be made accessible to those down-stream components of the KBS. This is achieved in an identical manner as described for those design drivers declared within the Agility dialog box. As can be seen below, the functions relating to the check boxes residing in each of dialog boxes is identical in format to that described for the Agility dialog box.

*Supersonic*

```
void __fastcall Tsupersonic::thin_section_cbClick(TObject *Sender)
{
int r;
r = 1;
        if(thin_section_cb->State ==cbChecked)
        {
        supersonic_ir_select.supersonic_thin_section_ir[0] = r;
        }
```

```
        if(thin_section_cb->State ==cbUnchecked)
        {
        supersonic_ir_select.supersonic_thin_section_ir[0] = '\0';
        }
}
```

*Subsonic*

```
void __fastcall Tsubsonic::thin_section_cbClick(TObject *Sender)
{
int r;
r = 1;
        if(thin_section_cb->State == cbChecked)
        {
        subsonic_ir_select.subsonic_thin_section_ir[0] = r;
        }

        if(thin_section_cb->State == cbUnchecked)
        {
        subsonic_ir_select.subsonic_thin_section_ir[0] = '\0';
        }
}
```

The code that relates to the operation of the radio buttons residing in the Research and Development Input dialog box as described in section 9.2.2.2 of Chapter 9 is given below. As with the Agility, Supersonic and Subsonic dialog boxes the level of Research and Development (R&D) selected must be made visible to other down-stream components of the KBS. The code that supports this capability is given below.

*R&D Level 1*

```
void __fastcall Tr_d_input::rd1_rbClick(TObject *Sender)
{
int a;
a = 1;
res_and_dev.research_and_development[0] = a;
}
```

*R&D Level 2*

```
void __fastcall Tr_d_input::rd2_rbClick(TObject *Sender)
{
int b;
b = 2;
res_and_dev.research_and_development[0] = b;
}
```

*R&D Level3*

```
void __fastcall Tr_d_input::rd3_rbClick(TObject *Sender)
{
int c;
c = 3;
```

```
res_and_dev.research_and_development[0] = c;
}
```

It can be seen from the above functions that relate to the operation of the individual radio buttons that the code is uncomplicated. Taking the function that corresponds to operation of the radio button that reflects the selection of R&D Level 1 as an example. When this radio button is selected the appropriate function above is called. The function declares an integer variable *a* and assigns the value 1 to it. The value 1 assigned to the integer variable is then itself assigned to the array *research_and_development*. This array resides in the *structure* RES_AND_DEV_SELECTION which is declared in the header file *buffer.h* i.e., making the value assigned to the array *research_and_development* globally visible.

The code that supports the underlying operation of the design drivers residing in the Cost dialog box is identical in terms of mode of operation to that employed within the Agility, Supersonic, Subsonic and Core-Competences dialog boxes. For the sake of completeness, the code that relates to these design drivers is given below. However, the reader is referred to the discussions relating to the Agility, Supersonic, Subsonic and Core-Competences dialog boxes given above for an explanation of the operation of this code.

*Start-up*

```
void __fastcall Tcost_input::start_up_cbClick(TObject *Sender)
{
int a;
a = 1;
        if(start_up_cb->State == cbChecked)
        {
        cost_ir_select.startup_ir[0] = a;
        }

        if(start_up_cb->State == cbUnchecked)
        {
        cost_ir_select.startup_ir[0] = '\0';
        }
}
```

*Material*

```
void __fastcall Tcost_input::material_cbClick(TObject *Sender)
{
int b;
b = 2;
        if(material_cb->State == cbChecked)
        {
        cost_ir_select.materials_ir[0] = b;
        }

        if(material_cb->State == cbUnchecked)
        {
```

```
        cost_ir_select.materials_ir[0] = '\0';
        }
}
```

*Labour*

```
void __fastcall Tcost_input::labour_cbClick(TObject *Sender)
{
int c;
b = 3;
        if(labour_cb->State == cbChecked)
        {
        cost_ir_select.labour_ir[0] = c;
        }

        if(labour_cb->State == cbUnchecked)
        {
        cost_ir_select.labour_ir[0] = '\0';
        }
}
```

*Processing Time*

```
void __fastcall Tcost_input::processing_time_cbClick(TObject *Sender)
{
int d;
d = 4;
        if(processing_time_cb->State == cbChecked)
        {
        cost_ir_select.processing_time_ir[0] = d;
        }

        if(processing_time_cb->State == cbUnchecked)
        {
        cost_ir_select.processing_time_ir[0] = '\0';
        }
}
```

*Part Reduction*

```
void __fastcall Tcost_input::part_reduction_cbClick(TObject *Sender)
{
int e;
e = 5;
        if(part_reduction_cb->State == cbChecked)
        {
        cost_ir_select.part_reduction_ir[0] = e;
        }

        if(part_reduction_cb->State == cbUnchecked)
        {
        cost_ir_select.part_reduction_ir[0] = '\0';
        }
}
```

*Inspection*

```
void __fastcall Tcost_input::inspection_cbClick(TObject *Sender)
{
int f;
f = 6;
        if(inspection_cb->State == cbChecked)
        {
        cost_ir_select.inspection_ir[0] = f;
        }

        if(inspection_cb->State == cbUnchecked)
        {
        cost_ir_select.inspection_ir[0] = '\0';
        }
}
```

*Assembly*

```
void __fastcall Tcost_input::assembly_cbClick(TObject *Sender)
{
int g;
g = 7;
        if(assembly_cb->State == cbChecked)
        {
        cost_ir_select.assembly_ir[0] = g;
        }
        if(assembly_cb->State == cbUnchecked)
        {
        cost_ir_select.assembly_ir[0] = '\0';
        }
}
```

*One-off Production*

```
void __fastcall Tcost_input::one_off_rbClick(TObject *Sender)
{
int h;
h = 1;
cost_ir_select.production_ir[0] = h;
}
```

*Batch Production*

```
void __fastcall Tcost_input::batch_rbClick(TObject *Sender)
{
int i;
i = 2;
cost_ir_select.production_ir[0] = i;
}
```

*Mass Production*

```
void __fastcall Tcost_input::mass_rbClick(TObject *Sender)
{
```

```
int j;
j = 3;
cost_ir_select.production_ir[0] = j;
}
```

### C.3.2. The Best Structure

This section discusses the code supporting a range of operations performed by the Best Structure dialog box in process of deriving the 'best structure'.

#### C.3.2.1. Best Structure - Data Initialisation

In order for the Best Structure dialog box to function i.e., to declare the best structure from a manufacturing and structures perspective, it must be able to draw on information from all the components of the ICES KBS that have an input to make. This is achieved in two ways. Firstly, the Best Structure dialog box has access to the header file *buffer.h*. As such, it is able to gain access to the global values placed there by the operations carried out in other dialog boxes impacting on the KBS component of the ICES prototype as described in Chapter 9. Secondly, the Best Structure dialog box is able to access information via standard file handling e.g., in the context of the design driver importance ratings derived from the Design Driver Ranking Matrix Output dialog box.

To initiate the operation of the Best Structure dialog box and hence enable the KBS to derive the best structure the user should click with the cursor on the 'Best Structure ?' push button, see Figure 9.15. This command calls the function *Tb_structure* which sets in motion the codified operations which eventually result in the best structure being declared. In the context of this thesis it would be impractical to display this entire function *Tb_structure* here. Therefore, this discussion will take code segments from this function in a logical manner presenting the salient points to the reader. Thus, enabling the underlying functionality of the supporting code to be clearly understood.

As indicated above the Best Structure dialog box draws on information from all the KBS components that facilitate the derivation of the best structure i.e., via access to the global variables residing in the header file *buffer.h* and through standard file handling. With respect to drawing information in from the header file *buffer.h*, the file *best_structure.cpp* which supports the underlying operation of the Best Structure dialog box includes the header file *buffer.h* in its include list i.e., with the code *#include "buffer.h"*. Then with the code below, the file *best_structure.cpp* accesses the *structures* listed, these being initially declared in *buffer.h*. Each of these structure calls provides access to the global variable values placed in the header file *buffer.h* by dialog boxes operating in the ICES KBS.

```
extern COST_STRUCT_IR_SELECTIONS cost_ir_select;
extern AGILITY_STRUCT_IR_SELECTIONS agility_ir_select;
extern RES_AND_DEV_SELECTION res_and_dev;
extern QUESTION_RESPONSE kbs_questions;
extern SECONDARY_RULES revised_rules;
```

extern ACTIVIATE_SECONDARY_RULES initiate_rules

As discussed in section C.1, the *extern* keyword tells the compiler that the program is going to use a variable in one particular source file but the compiler will find the variable in question declared in another source file. With respect to the code given above, the include statement *#include "buffer.h"* tells the program where the variables residing in the *structures* adjacent to the *extern* keyword may be found. Thus, the file *best_structure.cpp* is able to access these external variable values.

Turning attention now to the method of drawing in information to the Best Structure dialog box through standard file handling. Referring to section C.2.3.3, it will be recalled that the design driver importance ratings which were derived in the Design Driver Matrix Output dialog box where written to file i.e., *data17.txt*. The code that permits these values to be read from file resides in the function *Tb_structure* i.e., this function is called when the 'Best Structure ?' push button is clicked on. The code that corresponds to this file reading operation is given below. It should be noted that the principle aspects of this code are discussed in section C.2.3.3.

```
ifstream ifs ("data17.txt" , ios::binary)
ifs>>min_weight_imp_rat; //minimum weight
ifs>>high_str_str_imp_rat; //high structural strength
ifs>>low_tor_imp_rat; //low torque
ifs>>thin_sect_imp_rat; //thin section
ifs>>one_off_imp_rat; //one-off production
ifs>>bat_imp_rat; //batch production
ifs>>mass_imp_rat; //mass production
ifs>>start_up_imp_rat; //start-up
ifs>>mat_imp_rat; //materials
ifs>>lab_imp_rat; //labour
ifs>>part_red_imp_rat; //part reduction
ifs>>pro_time_imp_rat; //processing time
ifs>>insp_imp_rat; //inspection
ifs>>ass_imp_rat; //assembly
```

When the Best Structure dialog box is selected from the KBS menu the file *best_structure.cpp* proceeds to initialise a range of variables that are declared within the file. The first variables to be initialised correspond to the design driver/rule correlation factors as discussed in section 7.2.4.1 of Chapter 7. The code that corresponds to these correlation factors is given below.

```
first_rule_design_driver_cf = 1;
second_rule_design_driver_cf = 0.8;
third_rule_design_driver_cf = 0.6;
fourth_rule_design_driver_cf = 0.4;
fifth_rule_design_driver_cf = 0.2;
```

These variables are declared as float variables in the header file of *best_structure.cpp* i.e. *best_structure.h*. As indicated above, the file *best_structure.cpp* is the file that provides all the underlying code supporting the Best Structure dialog box. Included in

this file is the function *Tb_structure* which is initiated by the 'Best Structure ?' push button.

As indicated in section 7.2.4.1 of Chapter 7 which discusses the ICES methodology, all the rules operating in the manufacturing and structures rule bases are able to impact on one or more of seven distinct sections. The reader is reminded that the concept behind defining these sections is that when a rule fires in the manufacturing and/or structures rule bases each rule scores a weighting in one or more of these seven sections. The weighted score for each rule is determined from a positive correlation between the rules residing in the rule bases and the design drivers. In terms of implementing the methodology as described in Chapter 7 in a codified format it was necessary to make every rule within the two rule bases a stand-alone entity. That is, the weighted score achieved by any rule for any structure type is assigned to its own specific float variable. These variables in addition to the design driver/rule correlation factor variables indicated above also need to be initialised. However, these variables are initialised to 0. As an example, the code below illustrates the initialisation of the float variables in the manufacturing rule base that correspond to the rules relating to the standard x-core structure.

```
pr_standxcore_manu_batch_tooling4 = 0;
pr_standxcore_manu_batch_materials4 = 0;
pr_standxcore_manu_batch_startup4 = 0;
pr_standxcore_manu_oneoff_tooling4 = 0;
pr_standxcore_manu_oneoff_materials4 = 0;
pr_standxcore_manu_oneoff_startup4 = 0;
pr_standxcore_manu_sheetthickness_processingtime4 = 0;
pr_standxcore_manu_sheetthickness_inspection4 = 0;
pr_standxcore_manu_sheetthickness_mat4 = 0;
pr_standxcore_manu_sheetthickness_lab4 = 0;
pr_standxcore_manu_sheetthickness_pr1 = 0;
pr_standxcore_manu_sheetthickness_pr3 = 0;
pr_standxcore_manu_sheetthickness_ass7 = 0;
pr_standxcore_spfdb_manu_batch_startup1 = 0;
pr_standxcore_spfdb_manu_batch_partreduction3 = 0;
pr_standxcore_spfdb_manu_oneoff_startup1 = 0;
pr_standxcore_spfdb_manu_oneoff_partreduction3 = 0;
```

As with the correlation factor variables given previously, these variables are declared as float variables in the header file *best structure.h*. It can be seen that the rule variables corresponding to the standard x-core structure are all assigned a number at the end of the variable name. This number indicates which of the seven sections, as described in section 7.2.4.1 of Chapter 7, the eventual scores attributed to these variables by the firing of the corresponding rules will be assigned to.

### C.3.2.2 Best Structure - Checking for Manufacturing and R&D Input

As indicated in section C.3.2.1, the operation of the 'Best Structure ?' push button calls the function *Tb_structure* which resides in the file *best_structure.cpp*. After this function has read the design driver importance ratings from file it checks to see if the

user has selected a level of R&D and a method of manufacture. As explained in section C.3.1 when discussing the meta rule base, clicking on the radio buttons in the Research and Development Input and Cost dialog boxes causes numerical values to be assigned to an array which is declared in a *structure* in the header file *buffer.h*. The function *Tb_structure* which has access to values stored in *buffer.h* checks to see that there are values assigned relating to R&D and manufacture. This operation is performed with the following code.

*Production*

```
if(cost_ir_select.production_ir[0] == 0)
{
        manu_warn->ShowModal();
}
```

*R&D*

```
if(res_and_dev.research_and_development[0] == 0)
{
        no_res_dev->ShowModal();
}
```

The values in the arrays *production_ir* and *research_and_development* residing in the *structures* COST_STRUCT_IR_SELECTIONS and RES_AND_DEV_SELECTION respectively, which are declared in the header file *buffer.h*, are assigned the default value 0. When a radio button in the Research and Development Input or Cost dialog box is selected by the user this default value of 0 is over-written. It can be seen from the above code that the *if* conditional statements compare the values residing in the *structure* arrays with 0. If either of these conditional *if* statements holds true then the conditional statement is entered and the system initiates the display of a dialog message box telling the user that he or she has neglected to select a method of manufacture or has not considered the input of R&D. Figure 9.16 shows the Manufacturing and Research and Development Warning dialog message boxes. For an explanation of how the code that facilitates the displaying of these two dialog boxes works the reader is referred to section C.2.2 which discusses how the Structure Identification dialog box is displayed. This being an identical format to the method used to display the Manufacturing and Research and Development Warning dialog boxes.

### C.3.2.3. Best Structure - Codified examples of Rules in the Manufacturing and Structures Rule Bases

With reference to section 9.2.2.4 of Chapter 9, this section explains the underlying code that supports the operation of rules in the manufacturing and structures rule bases. Assuming that the user has made all the appropriate system inputs and clicked on the 'Best Structure?' push button in the Best Structure dialog box the ICES KBS proceeds to fire the rules residing in the function *Tb_structure*. The reader is referred to section C.3.2.1 for more information relating to the function *Tb_structure*. Clearly, with the large number of rules residing in the function *Tb_structure* it is not possible to discuss the operation of each one individually. However, the sample of rules presented in this discussion should provide the reader with a thorough understanding

of how the rules corresponding to the manufacturing and structures rules bases residing in the function *Tb_structure* operate.

Before providing codified examples of the rules relating to the manufacturing and structures rule bases the reader is referred to section 7.2.4.1 of Chapter 7. As indicated in this section, each rule that fires achieves a weighted score. This score is based on multiplying a positive correlation factor by the design driver importance rating. The score is then assigned to one of seven distinct sections. It is important to note that every rule in the KBS is applied to every structural type in the system in order to eliminate bias. That is, if every rule where not applied to every structure there would be those structures more favoured than others. Now consider the cluster of rules in the segment of code below.

```
if(cost_ir_select.production_ir[0] == 3)
{
//mass production and SPF
pr_spf_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);
pr_standxcore_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);
pr_spacedxcore_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);
pr_accordion_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);
pr_cellular_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);

//mass production - materials
pr_spf_manu_mass_materials4 = (mat_imp_rat*first_rule_design_driver_cf);
pr_standxcore_manu_mass_materials4 = (mat_imp_rat*first_rule_design_driver_cf);
pr_spacedxcore_manu_mass_materials4                                    =
(mat_imp_rat*first_rule_design_driver_cf);
pr_accordion_manu_mass_materials4 = (mat_imp_rat*first_rule_design_driver_cf);
pr_cellular_manu_mass_materials4 = (mat_imp_rat*first_rule_design_driver_cf);

//mass production - start-up
pr_spf_manu_mass_startup4=(startup_imp_rat*first_rule_design_driver_cf);
pr_standxcore_manu_mass_startup4=(startup_imp_rat*first_rule_design_driver_cf);
pr_spacedxcore_manu_mass_startup4=(startup_imp_rat*first_rule_design_driver_cf);
pr_accordion_manu_mass_startup4=(startup_imp_rat*first_rule_design_driver_cf);
pr_cellular_manu_mass_startup4 = (startup_imp_rat*first_rule_design_driver_cf);
}
```

The above code segment comes from the manufacturing rule base. It can be seen that the code is somewhat more than just a traditional IF THEN rule. The traditional *if* conditional statement in fact bounds a cluster of rules. The *if* conditional statement *if(cost_ir_select.production_ir[0] == 3)* says if mass production is the preferred method of manufacture then proceed i.e., enter the conditional statement. The *production_ir* array is declared in the header file *buffer.h*. The value 3 may be assigned to this array by selecting the radio button corresponding to mass production in the Cost dialog box.

Now looking inside the conditional statement i.e., assuming mass production is the preferred method of manufacture. It can be seen that there are three sections within

the if conditional statement, each with five rules, making fifteen in all. It should be noted that these sections are artificial in that there is no practical need to segment the code up in this manner, it just facilitates clarity of understanding. Taking the first segment of five rules.

*SPF titanium*
pr_spf_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);

*Standard x-core*
pr_standxcore_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);

*Spaced x-core*
pr_spacedxcore_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);

*Accordion core*
pr_accordion_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);

*Cellular core*
pr_cellular_manu_mass_tooling4 = (mass_imp_rat*first_rule_design_driver_cf);

These five rules are identical except that the result is assigned to the float variables corresponding to the five super plastic formed (SPF) and super plastic formed diffusion bonded (SPF/DB) structures supported within the ICES KBS. Taking the rule relating to standard x-core; what this rule says, is that where mass production is required and mass production is the design driver i.e., *mass_imp_rat*, there is a very strong correlation i.e., *first_rule_design_driver_cf*, between standard x-core and its capability to be manufactured in a mass production environment. The value assigned to the correlation factor is multiplied by the mass production design driver. The value obtained from this operation is then assigned to the float variable *pr_standxcore_manu_mass_tooling4*. The variable is assigned to the fourth section of the seven sections referred to above and in section 7.2.4.1 of Chapter 7. Turning attention now to the second segment of rules in the *if* conditional statement.

*SPF titanium*
pr_spf_manu_mass_materials4 = (mat_imp_rat*first_rule_design_driver_cf);

*Standard x-core*
pr_standxcore_manu_mass_materials4 = (mat_imp_rat*first_rule_design_driver_cf);

*Spaced x-core*
pr_spacedxcore_manu_mass_materials4 =
(mat_imp_rat*first_rule_design_driver_cf);

*Accordion core*
pr_accordion_manu_mass_materials4 = (mat_imp_rat*first_rule_design_driver_cf);

*Cellular core*
pr_cellular_manu_mass_materials4 = (mat_imp_rat*first_rule_design_driver_cf);

In common with the five rules discussed previously, the five above are also identical except that the result is assigned to five different float variables which correspond to the five SPF and SPF/DB titanium structures. Again taking the rule relating to standard x-core as an example. What this rule says, is that where mass production is required and material is the design driver i.e., *mat_imp_rat*, there is a very strong correlation i.e., *first_rule_design_driver_cf*, between stand x-core and its ability to be manufactured in a mass production environment and the purchase of materials in large quantities. In a similar manner to the previous example, the value assigned to the correlation factor is multiplied by the material design driver. The output from this operation being assigned to the float variable *pr_standxcore_manu_mass_materials4*. Looking now at the final segment of rules in the *if* conditional statement.

*SPF titanium*
pr_spf_manu_mass_startup4=(startup_imp_rat*first_rule_design_driver_cf);

*Standard x-core*
pr_standxcore_manu_mass_startup4=(startup_imp_rat*first_rule_design_driver_cf);

*Spaced x-core*
pr_spacedxcore_manu_mass_startup4=(startup_imp_rat*first_rule_design_driver_cf);

*Accordion core*
pr_accordion_manu_mass_startup4=(startup_imp_rat*first_rule_design_driver_cf);

*Cellular core*
pr_cellular_manu_mass_startup4 = (startup_imp_rat*first_rule_design_driver_cf);

The format of this segment of rules is identical to that described for the other two segments. Once again taking the rule relating to standard x-core. What this rule says, is that where mass production is required and start-up costs is the design driver i.e., *startup_imp_rat*, there is a very strong correlation between standard x-core and its ability to be manufactured in a mass production environment and start-up costs. As with the previous two examples, the value assigned to the correlation factor is multiplied by the start-up costs design driver. The value obtained from this exercise is then assigned to the float variable *pr_standxcore_manu_mass_startup4*.

It is considered that by studying the operation of the above cluster of rules residing in the manufacturing rule base the reader may have begun to gain appreciation of how rules in the KBS are constructed. In order to further enhance the reader's understanding, consider the following rules taken from the structures rule base.

```
if(kbs_questions.internal_section[0] == 1)
{
        if(((kbs_questions.load_path[0] == 1)&&
          (agility_ir_select.agility_minimum_weight_ir[0] == 1))&&
          ((cost_ir_select.production_ir[0] == 1)||
          (cost_ir_select.production_ir[0] == 2)||
```

```
 (cost_ir_select.production_ir[0] == 3)))
 {
 pr_standxcore_corealignment_struct_mw2=
 (min_weight_imp_rat*fourth_rule_design_driver_cf);
 pr_spacedxcore_corealignment_struct_mw2=
 (min_weight_imp_rat*fourth_rule_design_driver_cf);
 }

if(((kbs_questions.load_path[0] == 1)&&
 (agility_ir_select.materials_weight_ir[0] == 2))&&
 ((cost_ir_select.production_ir[0] == 1)||
 (cost_ir_select.production_ir[0] == 2)||
 (cost_ir_select.production_ir[0] == 3)))
 {
 pr_standxcore_corealignment_struct_mat4=
 (mat_imp_rat*fourth_rule_design_driver_cf);
 pr_spacedxcore_corealignment_struct_mat4=
 (mat_imp_rat*fourth_rule_design_driver_cf);
 }

if(((kbs_questions.load_path[0] == 1)&&
 (agility_ir_select.agility_high_structural_strength_ir[0] == 2))&&
 ((cost_ir_select.production_ir[0] == 1)||
 (cost_ir_select.production_ir[0] == 2)||
 (cost_ir_select.production_ir[0] == 3)))
 {
 pr_standxcore_corealignment_struct_hss2=
 (high_str_str_imp*fourth_rule_design_driver_cf);
pr_spacedxcore_corealignment_struct_hss2=
(high_str_str_imp*fourth_rule_design_driver_cf);
 }
}
```

As with the prior example, an *if* conditional statement i.e., *if(kbs_questions.internal_section[0] == 1)*, bounds three clusters of rules. These three clusters of rules are themselves bound by three further *if* conditional statements. The outer *if* conditional statement says, if the structure has an internal section then enter the conditional statement. The array *internal_section* is declared in the header file *buffer.h*. The value 1 is assigned to this array if the user enters 'y' in the edit box in the Questions dialog box corresponding to the question, "Does the structure have an enclosed load bearing internal section?". Turning attention now to the first of the three nested *if* conditional statements and associated rules.

*Minimum weight*

```
if(((kbs_questions.load_path[0] == 1)&&
        (agility_ir_select.agility_minimum_weight_ir[0] == 1))&&
        ((cost_ir_select.production_ir[0] == 1)||
        (cost_ir_select.production_ir[0] == 2)||
        (cost_ir_select.production_ir[0] == 3)))
```

```
{
pr_standxcore_corealignment_struct_mw2=
(min_weight_imp_rat*fourth_rule_design_driver_cf);
pr_spacedxcore_corealignment_struct_mw2=
(min_weight_imp_rat*fourth_rule_design_driver_cf);
}
```

The *if* conditional statement for the above rules says, if the structure has to support out of plane loads and the minimum weight design driver is selected, and one-off production or batch production or mass production is selected then enter the conditional statement. The two rules in the conditional statement are identical except that the result is assigned to two different float variables. These float variables correspond to the SPF/DB structures standard x-core and spaced x-core.

Taking the rule relating to spaced x-core as an example and embracing both sets of conditional statement. What this rule says, is where the structure has an internal section and has to support out of plane loads, and an appropriate method of manufacture has been selected, and minimum weight is the design driver; there is little positive correlation between the rule requirement and the minimum weight design driver. The reason for this poor correlation is that spaced x-core is a poor structure in terms of carrying out of plane loads and there is a weight penalty to be taken account of. In addition, it can be seen that the numerical score assigned to the float variable corresponding spaced x-core i.e., *pr_spacedxcore_corealignment_struct_mw2*, is assigned to section two of the seven sections discussed in section 7.2.4.1 of Chapter 7. Section two relates to the structure's load carrying capability. Looking now at the second of the nested *if* conditional statements.

*Material*

```
if(((kbs_questions.load_path[0] == 1)&&
        (agility_ir_select.materials_weight_ir[0] == 2))&&
        ((cost_ir_select.production_ir[0] == 1)||
        (cost_ir_select.production_ir[0] == 2)||
        (cost_ir_select.production_ir[0] == 3)))
        {
        pr_standxcore_corealignment_struct_mat4=
        (mat_imp_rat*fourth_rule_design_driver_cf);
        pr_spacedxcore_corealignment_struct_mat4=
        (mat_imp_rat*fourth_rule_design_driver_cf);
        }
```

The *if* conditional statement above is very similar to the previous *if* conditional statement. The only difference being that this one embraces the material design driver rather than the minimum weight design driver. Again, taking on board both conditional statements, the rule relating to the structure spaced x-core says, where the structure has an internal section and has to support out of plane loads, and an appropriate method of manufacture has been selected, and material is the design driver, there is little positive correlation between the rule requirement and the material design driver. The output score assigned to the float variable which corresponds to the spaced x-core structure i.e., *pr_spacedxcore_corealignment_struct_mat4*, registers in

section four of the seven sections. Section four relates to fabrication. The reason for the poor correlation is related to spaced x-core's inability to carry out-of-plane loads efficiently. That is, as spaced x-core cannot transfer out-of-plane loads through the core it has to do so through the skin. As such, there is a requirement for thicker skins and an increase in material usage. Turning attention now to the final nested if conditional statement from the above code.

*High structural strength*

```
if(((kbs_questions.load_path[0] == 1)&&
        (agility_ir_select.agility_high_structural_strength_ir[0] == 2))&&
        ((cost_ir_select.production_ir[0] == 1)||
        (cost_ir_select.production_ir[0] == 2)||
        (cost_ir_select.production_ir[0] == 3)))
        {
        pr_standxcore_corealignment_struct_hss2=
        (high_str_str_imp*fourth_rule_design_driver_cf);
      pr_spacedxcore_corealignment_struct_hss2=
      (high_str_str_imp*fourth_rule_design_driver_cf);
      }
```

As can be seen from the above code, there is a marked similarity with the two examples given previously. The difference between this example and the other two is that this one embraces the high structural strength design driver rather than the minimum weight or material design drivers. At the risk of repetition, taking on board both conditional statements, the rule relating to the structure spaced x-core says, where the structure has an internal section and has to support out of plane loads, and an appropriate method of manufacture has been selected, and high structural strength is the design driver, there is little positive correlation between the rule requirement and the high structural strength design driver. The output score assigned to the float variable corresponding to the spaced x-core structure i.e., *pr_spacedxcore_corealignment_struct_hss2*, registers in section two, which relates to load carrying capability. The reason for the poor correlation i.e., *fourth_rule_design_driver_cf* assigning a correlation factor of 0.4, is again related to spaced x-core's inability to carry out-of-plane loads efficiently. That is, as spaced x-core cannot transfer out-of-plane loads through the core there is a necessity to adopt thicker skins. The use of thicker skins in order to satisfactorily carry out of plane loads is structurally inefficient in the context of the high structural strength design driver when compared to structures which can support ribs or other comparable medium.

### C.3.2.4. Best Structure - Determining the Best Structure from Rules Fired in the Manufacturing and Structures Rule Bases

Section C.3.2.3 explains the code that supports the operation of rules in the manufacturing and structures rule bases. This section follows on from section C.3.2.3 and explains the codified operations necessary to enable the output of rules fired in the structures and manufacturing rule base to be used to determine the best structure. As indicated in section C.3.2.1, the output of every rule operating in the KBS is assigned an individual float variable. The next step to enable the KBS to determine the best structure is to sum all these float variables in the context of both the manufacturing

and structures rule bases and the individual structural types. The code segment below sums the float variables relating to the structure standard x-core for both the manufacturing and structures rule bases.

*Standard x-core manufacturing rules total*

```
pr_stand_x_core_manu_total =
(pr_standxcore_manu_batch_tooling4+
 pr_standxcore_manu_batch_materials4+
 pr_standxcore_manu_batch_startup4+
 pr_standxcore_manu_oneoff_tooling4+
 pr_standxcore_manu_oneoff_materials4+
 pr_standxcore_manu_oneoff_startup4+
pr_standxcore_manu_sheetthickness_startup4+
pr_standxcore_manu_sheetthickness_processingtime4+
pr_standxcore_manu_sheetthickness_inspection4+
pr_standxcore_manu_sheetthickness_mat4+
pr_standxcore_manu_sheetthickness_lab4+
pr_standxcore_manu_sheetthickness_pr1+
pr_standxcore_manu_sheetthickness_pr3+
pr_standxcore_manu_sheetthickness_ass7+
pr_standxcore_spfdb_manu_batch_startup1+
pr_standxcore_spfdb_manu_batch_partreduction3+
pr_standxcore_spfdb_manu_oneoff_startup1+
pr_standxcore_spfdb_manu_oneoff_partreduction3+
pr_standxcore_manu_mass_tooling4+
 pr_standxcore_manu_mass_materials4+
 pr_standxcore_manu_mass_startup4+
 pr_standxcore_spfdb_manu_mass_startup1+
 pr_standxcore_spfdb_manu_mass_partreduction3);
```

*Standard x-core structures rules total*

```
pr_stand_x_core_struct_total =
(pr_standxcore_struct_sheetthickness_mw2+
 pr_standxcore_corealignment_struct_mw2+
 pr_standxcore_struct_corealignment_struct_mat4+
 pr_standxcore_struct_corealignment_struct_hss2+
 pr_standxcore_spfdb_struct_homogenous_mw2+
 pr_standxcore_spfdb_struct_homogenous_hss2+
 pr_standxcore_spfdb_struct_homogenous_pr1+
 pr_standxcore_spfdb_struct_homogenous_pr3+
 pr_standxcore_spfdb_struct_homogenous_ts4);
```

After the float variables corresponding to each rule relating to each structure in the manufacturing and structures rule bases have been totalled in the manner indicated above for the standard x-core structure, a comparison is made between the totals for each structure. The purpose of this comparison is to determine whether any two or more structures have the same score totals. To ensure that it is possible to rank structures in descending order of preference it is vital that all the scores be different. This requirement for the structure score totals to be different is not just important in ensuring the operation of the methodology as outlined in Chapter 7 but also in terms

of the operation of the visual components of C++ Builder. It can be seen from the Best Structure dialog box as illustrated in Figure 9.15 that the structures corresponding to both the manufacturing and structures rule bases are ranked in descending order of preference within two list boxes. Clearly, to be able to rank the structures in such a manner there has to be a numerical difference between the structure score totals.

The likelihood in all circumstances of achieving a situation where all the score totals relating to each structure in each rule base are different are remote. This is because many of the structures under consideration have more aspects in common than they have differences. This is particularly true of the SPF/DB structures i.e., standard x-core, spaced x-core, accordion core and cellular core. In order to overcome the potential problem of having two or more structures with the same score totals it was necessary to allow for the worst case scenario; where all the structures have the same score totals. In this situation, in order to differentiate between the structures it was necessary to add a different minute floating point value to each of the structures embraced by the KBS. It is important to note that the floating point value assigned to each of the structures is a couple of magnitudes smaller than those numerical values obtained when rules are fired. As such, these artificially assigned scores have no impact in the further operation of the KBS beyond their intended purpose.

Clearly, no matter how small the value added to the structure score totals the effect is to artificially rank the structures. It was therefore necessary to decide in the event that all the structures have the same score how to rank them. It was decided to rank the structures according to BAe MA&A's experience with the structure. As such, spaced x-core was placed first followed by standard x-core. Spaced x-core is used on the Eurofighter foreplane while standard x-core is identical to spaced x-core except for the spacing of the cores. Accordion core has been used successfully on the Tornado and as such placed third in the ranking order. Whilst BAe MA&A has little experience with cellular core it seemed appropriate to keep all the SPF/DB structures together and as such this structure was placed fourth. As SPF titanium is clearly closely related to SPF/DB structures it was logical to place this structure type fifth in the order. The remaining structure types were CFC and traditional stressed skin these were placed sixth and seventh respectively. Clearly, the reader may disagree with the order of these structure types but in terms of the operation of the KBS it makes no difference as the numerical values assigned to these structure scores are truly minute. The only purpose they serve is error catching. The code that resides in the function *Tb_structure* that provides this artificial ranking in the event that all the structure score totals are the same for the manufacturing rule base is given below.

```
const float U = 0.001;
const float V = 0.002;
const float W = 0.003;
const float X = 0.004;
const float Y = 0.005;
const float Z = 0.006;

if(audit_trail_best_struct.equal_best_structure[0] == 0)
{
        Spaced x-core
```

```
if(pr_spaced_x_core_manu_total == pr_spf_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total+Z);
}

if(pr_spaced_x_core_manu_total == pr_stand_x_core_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total+Z);
}

if(pr_spaced_x_core_manu_total == pr_accordion_core_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total+Z);
}

if(pr_spaced_x_core_manu_total == pr_cellular_core_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total+Z);
}

if(pr_spaced_x_core_manu_total == pr_trad_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total+Z);
}

if(pr_spaced_x_core_manu_total == pr_cfc_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total+Z);
}
```

*Standard x-core*

```
if(pr_stand_x_core_manu_total == pr_spf_manu_total)
{
pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total+Y);
}

if(pr_stand_x_core_manu_total == pr_accordion_core_manu_total)
{
pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total+Y);
}

if(pr_stand_x_core_manu_total == pr_cellular_core_manu_total)
{
pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total+Y);
}

if(pr_stand_x_core_manu_total == pr_trad_manu_total)
{
pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total+Y);
```

```
}

if(pr_stand_x_core_manu_total == pr_cfc_manu_total)
{
pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total+Y);
}
```

*Accordion core*

```
if(pr_accordion_core_manu_total == pr_spf_manu_total)
{
pr_accordion_core_manu_total = (pr_accordion_core_manu_total+X);
}

if(pr_accordion_core_manu_total == pr_cellular_core_manu_total)
{
pr_accordion_core_manu_total = (pr_accordion_core_manu_total+X);
}

if(pr_accordion_core_manu_total == pr_trad_manu_total)
{
pr_accordion_core_manu_total = (pr_accordion_core_manu_total+X);
}

if(pr_accordion_core_manu_total == pr_cfc_manu_total)
{
pr_accordion_core_manu_total = (pr_accordion_core_manu_total+X);
}
```

*Cellular core*

```
if(pr_cellular_core_manu_total == pr_spf_manu_total)
{
pr_cellular_core_manu_total = (pr_cellular_core_manu_total+W);
}

if(pr_cellular_core_manu_total == pr_trad_manu_total)
{
pr_cellular_core_manu_total = (pr_cellular_core_manu_total+W);
}

if(pr_cellular_core_manu_total == pr_cfc_manu_total)
{
pr_cellular_core_manu_total = (pr_cellular_core_manu_total+W);
}
```

*SPF titanium*

```
if(pr_spf_manu_total == pr_trad_manu_total)
{
pr_spf_manu_total = (pr_spf_manu_total + V);
}
```

```
if(pr_spf_manu_total == pr_cfc_manu_total)
{
pr_spf_manu_total = (pr_spf_manu_total + V);
}

Carbon fibre composite
if(pr_cfc_manu_total == pr_trad_manu_total)
{
pr_cfc_manu_total = (pr_cfc_manu_total + U);
}
```

It can be seen at the beginning of this code segment that six float variables are declared i.e., U, V, W, X, Y, and Z. These float variables are assigned values ranging from 0.001 to 0.006. As indicated above, these values where intentionally made very small so they would have no impact on the KBS operation other than what is intended.

It can be seen that the code leads on from declaring these float variables to compare the score total obtained for spaced x-core with the score totals for every other structure embraced by the KBS. The code employs standard *if* conditional statements to do this. If at any juncture the score total for spaced x-core is found to be equal to that of any other structure then the spaced x-core total is increased by 0.006 i.e., *pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total+Z)*. After the spaced x-core score total has been compared against all other structure score totals, it is then the turn of standard x-core to be compared against all remaining structure totals i.e. all structures except spaced x-core. However, this time, where standard x-core is found to have the same score total as another structure, the standard x-core score total is increased by only 0.005 i.e., *pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total+Y)*. The process continues in this manner for all the remaining structures contained in the KBS but the level by which each structures' score is artificially increased gradually diminishes depending on where they are placed in the artificial ranking order.

As indicated above, the example code provided relates to the manufacturing rule base. It should be noted that the code that provides the same capability in the structures rule base is almost identical. The only significant difference being the structure type variable names and as such, its operation is not commented upon here.

A further use for the method of artificially ranking structures relates to the operation of the Best Structure dialog box and in particular the firing of secondary rules. Before continuing, it should be noted that the operation of secondary rules in the context of the ICES prototype implemented in C++ Builder is discussed in section 9.2.2.4 of Chapter 9 and section C.3.2.7 of this appendix. The concept of secondary rules was introduced in section 7.2.4.2 of Chapter 7. As indicated in section 9.2.2.4, the preferred structural type for both the manufacturing and structures rule bases is placed at the top of two list boxes. Only when these two structural types are in agreement will a best structural type be placed in the edit box designated to display the best structure i.e., below the 'Best Structure ?' push button as illustrated in Figure 9.15. As intermated in sections 7.2.4.1 and 7.2.4.2 of Chapter 7, it is unlikely that the

structures rule base and the manufacturing rule base will be in agreement with respect to what is the 'best structure'. When this is the case, the user selects the desired perspective, be it the structural perspective or the manufacturing perspective. The user may then fire secondary rules which relate to the desired structure residing in the opposing rule base. The firing of the secondary rules relating to this structure will have the effect of moving the structure towards the top of the list box in the Best Structure dialog box that relates to the opposing rule base.

Whilst the theory relating to the firing of secondary rules as described in Chapter 7 is relatively straight forward, there are implementation problems. The principle problem being that the firing of a secondary rules can impact on other structures than the one for whom the output of the secondary rule was initially intended e.g. the firing of a secondary rule directed at standard x-core may impact on all other SPF/DB structures. The result being that there possibly is no relative change in the ranking of structural types in the manufacturing and structures rule bases i.e., as displayed in the list boxes of the Best Structure dialog box. In order to over come this potential problem it is necessary to apply the artificial ranking method described above in a slightly different manner than previously described.

Consider a situation where the manufacturing and structures rule bases are not in agreement and there is no declared best structure. The user in this instance, prefers to take the structures perspective and standard x-core is the preferred structural type of this rule base i.e., the structures rule base. The user now moves to the opposing rule base, being the manufacturing rule base, and fires the necessary secondary rules with the intention of making standard x-core the preferred structure for the manufacturing rule base as well. When this is achieved this will facilitate the Best Structure dialog box being able to present a best structure to the user i.e., being standard x-core. However, as already indicated, it is conceivable that the firing of secondary rules relating to the structure standard x-core may also impact on other structures in the manufacturing rule base and possibly also the structures rule base. The end result being, that there is not the intended relative change in the ranked order of importance of structures in the manufacturing rule base. In addition, there may also be unforeseen movement of structural types in the structures rule base. This problem is overcome with a two step process.

Firstly, when the secondary rules relating to a particular structure type e.g., standard x-core, are selected a numerical value is assigned to an array residing in a *structure* in the header file *buffer.h*. The value assigned to the array varies depending on which structural type the secondary rules fired correspond to. Secondly, when the 'Best Structure ?' push button is clicked on in the Best Structure dialog box and an attempt is made to achieve a match between the two rule bases preferred structures, the function *Tb_structure* checks to see if any secondary rules relating to any structure type have been fired. As indicated in section C.3.2.1, the file *best_structure.cpp* which supports the operation of the Best Structure dialog box is able to access all the structures declared in the header file *buffer.h*. The function *Tb_structure* which resides in the file *best_structure.cpp* checks to see if secondary rules have been fired with a range of variations on the following *if* conditional statement.

```
if(audit_trail_best_struct.equal_best_structure[0] == 1)
```

This *if* conditional statement compares the value residing in the array *equal_best_structure* which is declared in the *structure* BEST_STRUCTURE i.e., within the header file *buffer.h*, and which is pointed to by the *structure* pointer *audit_trail_best_struct* with the value 1. The value 1 indicates that secondary rules relating to the structure standard x-core have been fired. Numerical values other than 1 indicate that secondary rules relating to other structural types have been fired. If the *if* conditional statement test is passed then the *if* conditional statement itself is entered. Here standard x-core is compared with the structure score totals for every other structure embraced by the manufacturing rule base of the KBS. If the standard x-core score total is found to be equal with any other structure score then the score for standard x-core is incremented by 0.006. Thus, this ensures that while the firing of secondary rules relating to standard x-core may well impact on other structural types, it is only the standard x-core structure that improves its relative ranking in respect of other structures in the rule base. As can be seen from the code below, the method of comparing standard x-core with other structural types and subsequently incrementing the standard x-core score total is very similar to the method described previously for ensuring that no two structural types possess the same score totals.

```
if(audit_trail_best_struct.equal_best_structure[0] == 1)
{
        Standard x-core
        if(pr_stand_x_core_manu_total == pr_spf_manu_total)
        {
        pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total +Z);
        }

        if(pr_stand_x_core_manu_total == pr_spaced_x_core_manu_total)
        {
        pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total +Z);
        }

        if(pr_stand_x_core_manu_total == pr_accordion_core_manu_total)
        {
        pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total +Z);
        }

        if(pr_stand_x_core_manu_total == pr_cellular_core_manu_total)
        {
        pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total +Z);
        }

        if(pr_stand_x_core_manu_total == pr_trad_manu_total)
        {
        pr_stand_x_core_manu_total = (pr_stand_x_core_manu_total + Z);
        }

        if(pr_stand_x_core_manu_total == pr_cfc_manu_total)
        {
```

```
pr_stand_x_core_manu_total  = (pr_stand_x_core_manu_total +Z);
}
```

*Spaced x-core*

```
if(pr_spaced_x_core_manu_total == pr_spf_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total +Y);
}

if(pr_spaced_x_core_manu_total == pr_accordion_core_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total +Y);
}

if(pr_spaced_x_core_manu_total == pr_cellular_core_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total +Y);
}

if(pr_spaced_x_core_manu_total == pr_trad_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total +Y);
}

if(pr_spaced_x_core_manu_total == pr_cfc_manu_total)
{
pr_spaced_x_core_manu_total = (pr_spaced_x_core_manu_total +Y);
}
```

*Accordion core*

```
if(pr_accordion_core_manu_total == pr_spf_manu_total)
{
pr_accordion_core_manu_total = (pr_accordion_core_manu_total +X);
}

if(pr_accordion_core_manu_total == pr_cellular_core_manu_total)
{
pr_accordion_core_manu_total = (pr_accordion_core_manu_total +X);
}

if(pr_accordion_core_manu_total == pr_trad_manu_total)
{
pr_accordion_core_manu_total = (pr_accordion_core_manu_total +X);
}

if(pr_accordion_core_manu_total == pr_cfc_manu_total)
{
pr_accordion_core_manu_total = (pr_accordion_core_manu_total +X);
}
```

*Cellular core*

```
if(pr_cellular_core_manu_total == pr_spf_manu_total)
{
pr_cellular_core_manu_total = (pr_cellular_core_manu_total +W);
}

if(pr_cellular_core_manu_total == pr_trad_manu_total)
{
pr_cellular_core_manu_total = (pr_cellular_core_manu_total +W);
}

if(pr_cellular_core_manu_total == pr_cfc_manu_total)
{
pr_cellular_core_manu_total = (pr_cellular_core_manu_total +W);
}
```

*SPF titanium*

```
if(pr_spf_manu_total == pr_trad_manu_total)
{
pr_spf_manu_total = (pr_spf_manu_total + V);
}

if(pr_spf_manu_total == pr_cfc_manu_total )
        {
        pr_spf_manu_total = (pr_spf_manu_total +V);
        }
```

*Carbon fibre composite*

```
        if(pr_cfc_manu_total == pr_trad_manu_total +U)
        {
        (pr_cfc_manu_total  = (pr_cfc_manu_total +U);
        }
}
```

As indicated above, it is quite possible for the firing of secondary rules aimed at altering the relative position of standard x-core in the manufacturing rule base to impact on the relative ranking of other structural types in the structures rule base i.e., particularly SPF/DB structural types. To overcome this potential problem, the code contained in the *if* conditional statement *if(audit_trail_best_struct.equal_best_structure[0] == 1)* above is expanded to embrace the artificial ranking of structures described earlier in this appendix. However, this artificial ranking of structures will be applied to the structures residing in the rule base corresponding to the preferred perspective i.e., the structures rule base in this instance. The purpose of applying this artificial ranking of structures is to ensure there is no relative move of structural types in the rule base which corresponds to the preferred structural perspective.

Having described how all the structural types in the manufacturing and structures rule bases are assured a unique numerical score total, it is now proposed to show how these totals are sorted and ranked in descending order and displayed in the two list boxes in the Best Structure dialog box i.e., when the user clicks on the 'Best Structure ?' push button.

Having obtained the score totals for each structural type in both the manufacturing and structures rule bases, it is necessary to sort them. This facilitates the system down stream being able to list the range of structures corresponding to these scores in descending order of preference in each list box of the Best Structure dialog box. The first step in sorting the structure score totals is to assign them to two arrays corresponding to manufacturing and structures.

*Manufacturing*

```
float arr[N] = {pr_spf_manu_total, pr_stand_x_core_manu_total,
                pr_spaced_x_core_manu_total, pr_accordion_core_manu_total,
                pr_cellular_core_manu_total, pr_trad_manu_total, pr_cfc_manu_total};
```

*Structures*

```
float brr[N] = {pr_spf_struct_total, pr_stand_x_core_struct_total,
                pr_spaced_x_core_struct, pr_accordion_core_struct_total,
                pr_cellular_core_struct_total, pr_trad_struct_total,
                pr_cfc_struct_total};
```

With the structure score totals assigned to the appropriate arrays corresponding to the manufacturing and structure rule bases, the following two calls *bsort(arr, N)* and *bsort(brr, N)* are made to the function *bsort*. The function bsort shown below, sorts the arrays using a variation of the bubble sort.

```
void __fastcall Tb_structure::bsort(float* ptr, int n)
{
int j, k;
        for(j=0; j<n-1; j++)
                for(k=j+1; k<n; k++)
                order(ptr+j, ptr+k);
}

void __fastcall Tb_structure::order(float* numb1, float* numb2)
{
        if(*numb1 > *numb2)
        {
        float temp = *numb1;
        *numb1 = *numb2;
        *numb2 = temp;
        }
}
```

The bubble sort sorting technique is a well known approach to sorting. It is not proposed to discuss its operation in this thesis. However, the interested reader who

requires further information is guided to reference 98.Before the sorted structure scores can be assigned to the appropriate list boxes it is first necessary to create a location in memory to place the contents of the list boxes in.

```
TStringList *ManuStringList = new TStringList;
TStringList *StructStringList = new TStringList;
```

The pointers **ManuStringList* and **StructStringList* point to the object *TStringList* and the terms *new TStringList* assign memory.

The method of assigning the sorted structure scores to the list boxes corresponding to the structures and manufacturing rule bases in the Best Structure dialog box is very straight forward. The arrays *arr* and *brr* contain seven numerically sorted structure score totals each. The highest score being located in the seventh array position i.e., *arr[7]* and *brr[7]*. The value residing in the seventh position in each array is compared with the structure score totals. The code supporting this process in the context of the manufacturing structure score totals is shown below.

```
if(arr[7] == pr_spf_manu_total)
(
ManuStringList->Add("SPF Titanium");
}

if(arr[7] == pr_stand_x_core_manu_total)
{
ManuStringList->Add("Standard X-core");
}

if(arr[7] == pr_spaced_x_core_manu_total)
{
ManuStringList->Add("Spaced X-core");
}

if(arr[7] == pr_accordion_core_manu_total)
{
ManuStringList->Add("Accordion Core");
}

if(arr[7] == pr_cellular_core_manu_total)
{
ManuStringList->Add("Cellular Core");
}

if(arr[7] == pr_trad_manu_total)
{
ManuStringList->Add("Trad. Stressed Skin");
}

if(arr[7] == pr_cfc_manu_total)
{
```

```
ManuStringList->Add("Carbon Fibre Composite");
}
```

It can be seen that the highest structure score total in the array *arr*, which corresponds to manufacturing, is compared with all the individual scores totals for the manufacturing rule base through the application of a series of *if* conditional statement tests. If the conditional statement test proves to be true then the statement itself is entered and the string corresponding to the structure type in question is placed in the memory location reserved for the manufacturing list box. Having identified the highest scoring structure the process now moves to the second highest structural score which is held at the sixth array position i.e., *arr[6]* and *brr[6]*. As with the seventh array position, the value residing in the sixth position in each array is compared with the structure score totals. Again, the code supporting this process in the context of the manufacturing structure score totals is shown below.

```
if(arr[6] == pr_spf_manu_total)
(
ManuStringList->Add("SPF Titanium");
}

if(arr[6] == pr_stand_x_core_manu_total)
{
ManuStringList->Add("Standard X-core");
}

if(arr[6] == pr_spaced_x_core_manu_total)
{
ManuStringList->Add("Spaced X-core");
}

if(arr[6] == pr_accordion_core_manu_total)
{
ManuStringList->Add("Accordion Core");
}

if(arr[6] == pr_cellular_core_manu_total)
{
ManuStringList->Add("Cellular Core");
}

if(arr[6] == pr_trad_manu_total)
{
ManuStringList->Add("Trad. Stressed Skin");
}

if(arr[6] == pr_cfc_manu_total)
{
ManuStringList->Add("Carbon Fibre Composite");
}
```

In an identical manner as described previously, *if* conditional statements are employed to determine the second highest scoring structure. When this structure is identified a string description of this structure is added to the memory location reserved for the manufacturing list box. This process is repeated for all seven structural types for both the *aar* and *brr* arrays. Once this procedure has been completed the structural types supported by the manufacturing and structures rule bases will be retained in descending order of preference in memory. All that now remains to be done is to output the strings relating to these structural types to the appropriate list box. This is achieved through the application of the following code.

```
ListBox1->Items=ManuStringList;
ListBox2->Items=StructStringList;
```

The terms *ListBox1* and *ListBox2* are the names by which the application recognises the two list boxes residing in the Best Structure dialog box. The term *Items* is a keyword. It enables the strings residing in memory corresponding to the structural types to be placed in order in the appropriate list box.

### C.3.2.5. Best Structure - Comparison between the Best Structure and R&D Input

This section explains the code that supports the operation of comparing the declared best structure with the level of R&D input. If there is a discrepancy between the selected level of R&D and the declared best structure then this issue is brought to the users attention. The process of codifying the comparison of the best structure with the selected level of R&D firstly requires that the best structure as declared in the best structure edit box be accessed and transformed into a format that facilitates comparison. The code that supports this operation is given below.

```
AnsiString s;
int Size = Edit1->GetTextLen(); //get length of string in Edit1
char *name = new char[++Size]; //allocate space for string
Edit1->GetTextBuf(name,Size); // puts Edit->Text into buffer
s=name;
delete name; //frees memory allocated to buffer
```

This code segment above is described in detail in section C.3.2; it takes the text residing in the best structure edit box i.e., the best structure, and assigns it to the string object *s*.

```
if(s == "SPF Titanium");
{
seven_structures.structural_range[0] = 1;
}

if(s == "Standard X-core");
{
seven_structures.structural_range[0] = 2;
}
```

```
if(s == "Spaced X-core");
{
seven_structures.structural_range[0] = 3;
}

if(s == "Accordion Core");
{
seven_structures.structural_range[0] = 4;
}
```

```
if(s == "Cellular Core");
{
seven_structures.structural_range[0] = 5;
}

if(s == "Traditional Stressed Skin");
{
seven_structures.structural_range[0] = 6;
}

if(s == "Carbon Fibre Composite");
{
seven_structures.structural_range[0] = 7;
}
```

Having assigned the string in the best structure edit box to the string object *s* the above code progressively works through a series of *if* conditional statements attempting to match the string object *s* with each structural type. When a match is achieved, the *if* conditional statement in question is entered and a numerical value ranging between 1 and 7 is assigned to the array *structural_range*. This array is declared in the *structure* STRUCTURE_TYPES which is declared in the header file *buffer.h*. Note that *seven_structures* is the pointer to the *structure* STRUCTURE_TYPES. The assigning of numerical values to the array *structural_range* now facilitates the comparison between the structural type declared as the best structure in the best structure edit box and the level of R&D selected. Consider the code below which relates to cellular core.

```
if(((seven_structures.structural_range[0] == 5)&&
(cost_ir_select.production_ir[0] != 0)&&
((res_and_dev.research_and_development[0] == 1)||
(res_and_dev.research_and_development[0] == 2)))
{
     research_development_warning->ShowModal();
}
```

The above code says if the structure in question is cellular core i.e., *seven_structures.structural_range[0] == 5*, and a method of manufacture is selected, and the level of R&D selected is either level one or two then show the Research and Development Warning message dialog box that relates to cellular core. Figure 9.20

shows a Research and Development Warning message dialog box which corresponds to the cellular core structure.

### C.3.2.6. Best Structure - KBS Link with the Case Base

This section discusses the codified link between the ICES KBS and case base. To provide a physical link between the KBS and the case base i.e., facilitate the case base being able to display the case(s) that contain the best structure, the seven section score totals relating to every structural type in both the structures and manufacturing rule base are added. These combined seven section score totals for each structural type are then assigned to a series of arrays residing in the *structure* KBS_OUTPUT. This *structure* is declared in the header file *buffer.h*. As an example consider the code below.

```
struct_and_manu_output.cfc_manu_struct_total1[0] =
(pr_cfc_manu_batch_startup1+pr_cfc_manu_oneoff_startup1+
 pr_cfc_manu_mass_startup1+pr_cfc_struct_pr1+
 pr_cfc_manu_sheetthickness_pr1);
```

The above code adds all the rule score totals for the manufacturing and structures rule bases together for section one of the seven sections corresponding to the carbon fibre composite (CFC) structural type. The total score attained is then assigned to the array *cfc_manu_struct_total1*, which as indicated above is declared in the *structure* KBS_OUTPUT. The term *struct_and_manu_output* is a pointer to this structure. The scores assigned to arrays residing in the *structure* KBS_OUTPUT can be seen by the ICES case base and used to determine the case(s) in the case base which have the best structure.

### C.3.2.7. Secondary Rules

This section describes how the code that supports the operation of secondary rules relating to cellular core in the manufacturing rule base works. It should be noted that the method described for cellular core secondary rules is identical for all other secondary rules supporting other structural types in both the manufacturing and structures rule bases.

The code that supports the operation of the Cellular Core Manufacturing Secondary Rules dialog box is contained in the file *cell_core_manu_sr.cpp*. When the Cellular Core Manufacturing Secondary Rules dialog box is activated it is necessary to inform the Best Structure dialog box that this is the case. This is achieved by the file *cell_core_manu_sr.cpp* assigning the numerical value 1 to an array that is declared in a *structure* within the header file *buffer.h* i.e., *initiate_rules.cellular_core_manufacturing[0]=1*. The value 1 when read by the file *best_structure.cpp* indicates that the Cellular Core Manufacturing Secondary Rules dialog box has been activated.

Having informed the Best Structure dialog box that the Cellular Core Manufacturing Secondary Rules dialog box has been activated the file *cell_core_manu_sr.cpp* proceeds to initialise the variables corresponding to the design driver/rule correlation factors as discussed in section 7.2.4.1 of Chapter 7. In

addition, the file *cell_core_manu_sr.cpp* utilises standard file handling to read the design driver importance ratings which were derived in the Design Driver Matrix Output dialog box and written to file i.e., *data17.txt*. The code that supports the initialising of the variables corresponding to the design driver/rule correlation factors and the standard file handling necessary to read the design driver importance ratings from the file data17.txt is provided in section C.3.2.1.

As can be seen from Figure 9.21, the Cellular Core Manufacturing Secondary Rules dialog box provides considerable text in the context of explaining what the user must do in order to improve the design driver/rule correlation. Clearly, the text presented will depend on the entries made up to this juncture in the KBS. As such, it is necessary to have the capability for the Cellular Core Manufacturing Secondary Rules dialog box only to display text that is relevant to the current design scenario and where appropriate be able to blank out other text. Consider the following code which relates to three scenarios where either batch, one-off or mass production is the chosen method of manufacture.

```
Batch production
if(cost_ir_select.production_ir[0] == 2)
{
tooling
Label1->Caption="Batch production is the minimal level of production";
Label2->Caption="necessary to justify the tooling costs of SPF. However";
Label3->Caption="mass production would permit the costs to be amortised";
Label4->Caption="over a longer production run. Switch to Mass Production?";

materials
Label5->Caption="If it is feasible to switch to mass production then";
Label6->Caption="this will facilitate the purchase of materials in even";
Label7->Caption="larger quantities and a significant cost saving has the";
Label8->Caption="potential to be made";

start-up costs
Label9->Caption="If it is feasible to switch to mass production then this";
Label10->Caption="will permit the start-up costs to be amortised over a";
Label11->Caption="longer production run";
}

One-off Production
if(cost_ir_select.production_ir[0] == 1)
{
tooling
Label1->Caption="If it is desirable to continue to use SPF titanium then";
Label2->Caption="consider the possibility of moving to batch production or";
Label3->Caption="mass production. This will permit the tooling costs to be";
Label4->Caption="amortised over a longer production run";

materials
Label5->Caption="It is not practical to buy materials in the small quantities";
```

```
Label6->Caption="required for one-off production. If however, production was";
Label7->Caption="switched to batch or mass production it would then be";
Label8->Caption="possible to justify buying materials in large quantities";
start-up costs
Label9->Caption="The start-up costs for one-off production are prohibitive.";
Label10->Caption="Is it possible to switch from one-off production to batch";
Label11->Caption="or even mass production?";
}

Mass production
if(cost_ir_select.production_ir[0] == 3)
{
tooling
Label1->Caption="";
Label2->Caption="Rule does";
Label3->Caption="not apply";
Label4->Caption="";

materials
Label5->Caption="";
Label6->Caption="Rule does";
Label7->Caption="not apply";
Label8->Caption="";

start-up costs
Label9->Caption="Rule does";
Label10->Caption="not apply";
Label11->Caption="";
}
```

It can be seen that the code embraced by each of the three *if* conditional statements above is very similar. Each *if* conditional statement performs a test to see which is the user's preferred method of manufacture. Referring to section C.3.1, which discusses the code supporting the meta rule base, the Cost dialog box requires the user to select a preferred method of manufacture. Depending on the method selected i.e., one-off, batch or mass production, a numerical value, ranging between 1 and 3 is assigned to the *production_ir* array which is declared in the header file *buffer.h*. Turning attention to the code above and taking the first *if* conditional statement as an example which relates to batch product i.e., *if(cost_ir_select.production_ir[0] == 2)*. Here a comparison is made between the value residing in the array *production_ir* which resides in the *structure* COST_STRUCT_IR_SELECTIONS declared in the header file *buffer.h* and the value 2. If the value assigned to the array *production_ir* does prove to equal 2 then batch production is indeed the user's preferred method of manufacture and the code embraced by the *if* conditional statement may be accessed.

Inside the *if* conditional statement *if(cost_ir_select.production_ir[0] == 2)*, there is code which assigns the secondary rule text relating to batch production and cellular core to the appropriate Label components that reside in the Cellular Core Manufacturing Secondary Rules dialog box. Taking the first line of this code i.e.,

*Label1->Caption="Batch production is the minimal level of production".* The term *Label1* is the name by which the application recognises a specific Label component. The keyword *Caption* provides the ability to assign the text string in the quotes to this Label component.

Clearly, depending on which mode of manufacture is selected there are three different possible textual outputs that can be fed to the Label components *Label1* to *Label11*. It can be seen that code encapsulated within the *if* conditional statement that relates to mass production i.e., *if(cost_ir_select.production_ir[0] == 3)*, the text output to the Label components informs the user that the particular secondary rule in question does not apply.

As can be seen from Figure 9.21, against every secondary rule there is a check box. The user who wishes to fire a particular secondary rule should click on the appropriate check box. The clicking on a check box will initiate a function in the file *cell_core_manu_sr.cpp* which corresponds to the particular secondary rule in question. Taking the check box from the secondary rule that corresponds to tooling i.e., the first rule in each of the three *if* conditional statements *Label1* to *Label4*, as an example. The code relating to this check box is given below.

```
void __fastcall Tcell_core_manu_second_rules::rule1_cbClick(TObject *Sender)
{

        if(cost_ir_select.production_ir[0] == 2)
        {
                if(rule1_cb->State==cbChecked)
                {
                revised_rules.sr_spf_manu_batch_tooling4[0] =
                (bat_imp_rat*first_rule_design_driver_cf);
                }
                if(rule1_cb->State==cbUnchecked)
                {
                revised_rules.sr_spf_manu_batch_tooling4[0] =0;
                }
}

if(cost_ir_select.production_ir[0] == 1)
{
                if(rule1_cb->State==cbChecked)
                {
                revised_rules.sr_spf_manu_batch_tooling4[0] =
                (one_off_imp_rat*first_rule_design_driver_cf);
                }

                if(rule1_cb->State==cbUnchecked)
                {
                revised_rules.sr_spf_manu_batch_tooling4[0] =0;
                }
}
```

}

The code for the above rules operates in a similar manner to that described in section C.3.2.3. However, these rules provide an improved positive correlation between the rule requirement and the design driver i.e., compared to when the rule was initially declared in the manufacturing rule base. The scores output by the firing of these rules are fed to arrays residing in the *structure* SECONDARY_RULES declared in the header file *buffer.h*.

When the 'Best Structure?' push button in the Best Structure dialog box is clicked on following the firing of secondary rules, the improved rule scores are fed into the system. Thus, permitting the cellular core structure to ascend the list box in the Best Structure dialog box corresponding to the manufacturing rule base. An example of some of the code that permits the secondary rules fired in the Cellular Core Manufacturing Secondary Rules dialog box to update initial rule scores is provided below.

```
if(initiate_rules.cellular_core_manufacturing[0] ==1)
{
        if(revised_rules.sr_spf_manu_batch_tooling4[0]>
          pr_cellular_manu_batch_tooling4)
        {
        pr_cellular_manu_batch_tooling4=
        revised_rules.sr_spf_manu_batch_tooling4[0];
        }

        if(revised_rules.sr_spf_manu_batch_materials4[0]>
          pr_cellular_manu_batch_materials4)
        {
        pr_cellular_manu_batch_materials4=
        revised_rules.sr_spf_manu_batch_materials4[0];
        }

        if(revised_rules.sr_spf_manu_batch_startup4[0]>
          pr_cellular_manu_batch_startup4)
        {
         pr_cellular_manu_batch_startup4=
        revised_rules.sr_spf_manu_batch_startup4[0];
        }
}
```

The segment of code above which resides in the function *Tb_structure*, which itself resides in the file *best_structure.cpp* first tests to see if the Cellular Core Manufacturing Secondary Rules dialog box has been activated. This is achieved through a conditional *if* statement test i.e., *if(initiate_rules.cellular_core_manufacturing[0] ==1)*. The file *best_structure.cpp* has access to the value assigned to the array *cellular_core_manufacturing* which is declared in the structure ACTIVIATE_SECONDARY_RULES residing in the header

file *buffer.h*. If the value assigned to the array *cellular_core_manufacturing* equals 1 then the *if* conditional statement can be entered.

The *if* conditional statement *if(initiate_rules.cellular_core_manufacturing[0] ==1)* supports further *if* conditional statements, three of which are provided above as examples. These *if* conditional statements compare the numerical scores achieved by the firing of secondary rules residing in the Cellular Core Manufacturing Secondary Rules dialog box with the existing scores of the rules residing in the manufacturing rule base corresponding to cellular core. If the score attained by a secondary rule is greater than its counterpart in the manufacturing rule base then the score for the secondary rule is substituted for the one currently residing in the manufacturing rule base. In this manner, the overall numerical score attributed to cellular core is incrementally improved until the structure becomes the preferred structure for the manufacturing rule base.

It should be noted that the description provided in this appendix of how secondary rules are fired in the context of the structure cellular core holds true for other structural types. In addition, the method described is equally applicable to the structures rule base as the manufacturing rule base.

## C.4. Case Base Operation

This section discusses the code that supports the operation of the ICES prototype's case base capability.

### C.4.1. Close

This section explains how the code supporting the Close menu option in the New Case window permits the user to close down window. The code that facilitates this operation is as follows:

```
void __fastcall Tnew_case::new_case_exitClick(TObject *Sender)
{
        new_case_table->Edit();
        new_case_table->Post();
        Close();
}
```

When the user clicks on the Close menu the above function is called. The *new_case_table* term is the name by which the application recognises the Paradox database table called Foreplane. This table is where all the fields relating to case base are declared. It is important to note that the Paradox database software package and C++ Builder utilise the same Borland database engine. As such, the link between C++ Builder and Paradox is seamless. When the above function is called the database table Foreplane is switched to edit mode i.e., *new_case_table->Edit()*. The function then writes any changes made to the case base to the Foreplane database table with the code *new_case_table->Post()*. Having written any changes to the case base to the database table the *Close()* command is called. This command ends the operation of the New Case window and causes it to close down.

### C.4.2. Insert

This section describes the code supporting the operations in the Insert menu of the New Case window.

### C.4.2.1. Insert menu options - Insert Record

This section explains how the code supporting the Insert Record menu option in the New Case window operates. The purpose of the Insert Record menu option is to create a new blank record. The code function that supports this operation is shown below.

```
void __fastcall Tnew_case::insert_recordClick(TObject *Sender)
{
new_case_table->Insert();
new_case_table->FieldByName("Max Pos Shear Load kN") ->AsInteger = 0;
new_case_table->FieldByName("Max Neg Shear Load kN") ->AsInteger = 0;
new_case_table->FieldByName("Max Pos Bend Load kNm") ->AsInteger = 0;
new_case_table->FieldByName("Max Neg Bend Load kNm") ->AsInteger = 0;
new_case_table->FieldByName("Max Pos Torque Load kNm") ->AsInteger = 0;
new_case_table->FieldByName("Max Neg Torque Load kNm") ->AsInteger = 0;
new_case_table->FieldByName("Tensile Yield Stress MPa") ->AsInteger = 0;
new_case_table->FieldByName("UTS MPa") ->AsInteger = 0;
new_case_table->FieldByName("Young's Modulus MPa") ->AsInteger = 0;
new_case_table->FieldByName("Density Mg/m3") ->AsInteger = 0;
new_case_table->FieldByName("Case Number") ->AsInteger = 0;
new_case_table->FieldByName("Structure type") ->AsString = 0;
DBImage1->Picture->LoadFromFile("c:\\engd\\icesgr~1\\blank.bmp");
DBImage2->Picture->LoadFromFile("c:\\engd\\icesgr~1\\blank.bmp");
DBImage3->Picture->LoadFromFile("c:\\engd\\icesgr~1\\blank.bmp");
new_case_table->Post();

//Re-numbering Records
new_case_table->Edit();
new-case_table->FieldByName("Case Number") ->AsInteger = 1;

while(!new_case_table->Eof)
{
case_number=new_case_table->FieldByName("Case Number") ->Value;
new_case_number = case_number+1;
new_case_table->Next();
new_case_table->Edit();
new_case_table->FieldByName("Case Number")->AsInteger = new_case_number;
}

while(!new_case_table->Bof)
{
new_case_table->Prior();
```

```
new_case_table->Edit();
}
new_case_table->Post();
}
```

Taking the code presented in the above function in sequence. The first line of code in this function i.e., *new_case_table->Insert()*, inserts a new blank design case record in the Foreplane database table. However, as it is likely that the user may not insert all the data into the new record at once it is necessary to ensure that the case base can still be queried without the system crashing. With this in mind, it necessary to initialise certain fields of the new case design record. This is what the next sixteen lines of code in the above function achieve. It can be seen that the value 0 is assigned to fields requiring a numerical input and an empty string is assigned to the one field that takes a string input. In addition, blank bitmap graphics are assigned to all the graphics fields of the new case record.

Following the initialisation of the new case record it is necessary that an identification number be assigned to the new record. This is what the code below, taken from the above function, achieves.

```
new_case_table->Edit();
new-case_table->FieldByName("Case Number") ->AsInteger = 1;
```

As indicated in section C.4.1, the term *new_case_table* is the name by which the application recognises the Foreplane database table where the fields relating to the case base are declared. The database table is first put into edit mode. Then the field in the Foreplane database table corresponding to *Case Number* is identified and the value 1 is assigned to it.

Clearly, prior to the introduction of the new case record there would be an existing design case with 1 as the identification number. It is therefore necessary when a new case record is added to the case base that the system automatically re-number the cases. The need to assign a unique identification number to design case records is necessary in the context of being able to later query the case base The code below, taken from the above function provides the means to re-number design case records residing in the case base.

```
while(!new_case_table->Eof)
{
case_number=new_case_table->FieldByName("Case Number") ->Value;
new_case_number = case_number+1;
new_case_table->Next();
new_case_table->Edit();
new_case_table->FieldByName("Case Number")->AsInteger = new_case_number;
}
```

The code starts with the newly inserted record and a *while* conditional statement is called i.e., *while(!new_case_table->Eof)*. This *while* conditional statement says, that while the end of the Foreplane table is not reached then continue i.e., access the

code encapsulated within the two braces. Within these two braces the numerical value assigned to the field *Case Number* is itself assigned to the integer variable *case_number*. The value assigned to *case_number* is then incremented by 1 and assigned to another integer variable *new_case_number*. The next line of code then moves the application on to the next record in the case base i.e., *new_case_table->Next()*. This next design record is then put in edit mode with the code *new_case_table->Edit()*. With the design record in edit mode, the integer value assigned to the integer variable *new_case_number* is assigned to the Foreplane table field *Case Number*. Thus, this particular design record is re-numbered. The process now repeats itself until the end of the Foreplane table is reached.

Having re-numbered the design case records residing in the case base. The design record displayed to the user will be the last record in the Foreplane database table. In terms of completeness and ease of understanding, it is desirable that the system returns to the first design record in the case base. This will be the newly inserted design record. The code below, again taken from the function provided above, facilitates this operation.

```
while(!new_case_table->Bof)
{
new_case_table->Prior();
new_case_table->Edit();
}
```

The *while* conditional statement says, that while the beginning of the Foreplane table is not reached then continue i.e., access the code encapsulated within the two braces. Within these two braces, the first line of code moves the application back to the record prior to the one where the application currently resides. This design record is then placed in edit mode and the process continues. The while loop terminates when the first design record in the Foreplane database table is reached.

The final operation that is necessary when inserting a new design record in the case base is to write any changes i.e., the new design record and identification number, to the Foreplane database table. This is achieved with the line of code *new_case_table->Post()*.

### C.4.2.2. Insertion of Graphics into a New Design Case Record

This section explains how graphics may be inserted into a new design case record. Taking the insertion of a graphic into the design case record field Aircraft Type as an example. The user should select the Aircraft Type Graphics menu option from the Insert menu in the New Case window. This will initiate the process which enables a bitmap picture to be inserted into the Aircraft Type field of the new design case record. The code that supports this operation is given below.

```
void __fastcall Tnew_case::aircraft_type_graphicsClick(TObject *Sender)
{
        Open Dialog->FileName = "";
        if(Open Dialog->Execute())
```

```
        {
        new_case_table->Edit();
        DBImage1->Picture->LoadFromFile(OpenDialog->FileName);
        new_case_table->Post();
        }
}
```

When the Aircraft Type Graphics menu option is selected the above function is called. The code *OpenDialog->FileName = ""* calls the Windows Open dialog box. By scrolling through the various directories and files presented in the Open dialog box the desired bitmap file may be accessed. This file is clicked on with the cursor. When the bitmap file is selected it is assigned to the file handler *FileName*. When the OK push button in the Open dialog box is selected the *if* conditional statement in the above function is entered i.e., *if(OpenDialog->Execute())*. Taking the code contained in the two braces in sequence, the Foreplane database table is first put into edit mode i.e., *new_case_table->Edit()*. With the Foreplane database table in edit mode the bitmap file assigned to the *FileName* file handler is itself assigned to the Aircraft Type field of the new design record. This is achieved with the code *DBImage1->Picture->LoadFromFile(OpenDialog->FileName)*. It should be noted that the term *DBImage1* is the name by which the application recognises the C++ Builder graphic component. Having assigned the bitmap to the appropriate field of the new design record this change is posted to the Foreplane database table i.e., *new_case_table->Post()*.

The method described for entering a bitmap into the Aircraft Type field of the case base is identical for the Engineering Drawing field using the Eng. Draw. menu option and the Structure Graphic field using the Structure Graphic menu option.

### C.4.3. Edit

This section explains how fields in a design case record may be edited. The Edit menu in the New Case dialog box enables the user to edit the fields of design cases residing in the case base. When the user clicks on the Edit menu command a pull-down menu is revealed which has a menu option corresponding to every field in the New Case window, see Figure 9.23. It should be noted that by default all cases in the case base are read only. Thus, to edit cases the user has to deliberately select the appropriate case field from the Edit menu. Clearly, there are times when it is desirable to be able to edit all the case fields at once. With this in mind, the first menu option in the Edit menu is Edit - All. This menu option enables any field of a case record to be edited. The code that supports the Edit - All menu option is given below.

```
void __fastcall Tnew_case::edit_allClick(TObject *Sender)
{
DBMemo1->ReadOnly=false;
DBMemo2->ReadOnly=false;
DBMemo3->ReadOnly=false;
DBMemo4->ReadOnly=false;
DBMemo5->ReadOnly=false;
DBEdit1->ReadOnly=false;
```

```
DBEdit2->ReadOnly=false;
DBEdit3->ReadOnly=false;
DBEdit4->ReadOnly=false;
DBEdit5->ReadOnly=false;
DBEdit6->ReadOnly=false;
DBEdit7->ReadOnly=false;
DBEdit8->ReadOnly=false;
DBEdit9->ReadOnly=false;
DBEdit10->ReadOnly=false;
DBEdit11->ReadOnly=false;
DBEdit12->ReadOnly=false;
DBEdit13->ReadOnly=false;
new_case_table->Edit();
new_case_table->FieldByName("Component Description")->
Value=DBMemo1->Text;
new_case_table->FieldByName("Manufacturing2")->
Value=DBMemo2->Text;
new_case_table->FieldByName("Structure Description")->
Value=DBMemo3->Text;
new_case_table->FieldByName("Risk")->
Value=DBMemo4->Text;
new_case_table->FieldByName("Defence Field")->
Value=DBMemo5->Text;
new_case_table->FieldByName("Structure Type")->
Value=DBEdit1->Text;
new_case_table->FieldByName("Component Title")->
Value=DBEdit2->Text;
new_case_table->FieldByName("Max Pos Shear Load kN")->
Value=DBEdit3->Text;
new_case_table->FieldByName("Max Neg Shear Load kN")->
Value=DBEdit4->Text;
new_case_table->FieldByName("Max Pos Bend Load kNm")->
Value=DBEdit5->Text;
new_case_table->FieldByName("Max Neg Bend Load kNm")->
Value=DBEdit6->Text;
new_case_table->FieldByName("Max Pos Torque Load kNm")->
Value=DBEdit7->Text;
new_case_table->FieldByName("Max Neg Torque Load kNm")->
Value=DBEdit8->Text;
new_case_table->FieldByName("Tensile Yield Stress MPa")->
Value=DBEdit9->Text;
new_case_table->FieldByName("UTS MPa")->
Value=DBEdit10->Text;
new_case_table->FieldByName("Young's Modulus MPa")->
Value=DBEdit11->Text;
new_case_table->FieldByName("Density Mg/m3")->
Value=DBEdit12->Text;
new_case_table->FieldByName("Case Number")->
Value=DBEdit13->Text;
```

```
new_case_table->Post();
}
```

Taking the above code in sequence, it can be seen that all the memo and edit box components have their *ReadOnly* attributes declared as false i.e., these components are recognised by the application by terms of the type DBMemo and DBEdit. Following the *ReadOnly* attributes being declared *false* the Foreplane database table is put into edit mode with the code *new_case_table->Edit()*. Once in edit mode, each field in the design case record is assigned it's component name, be it DBMemo or DBEdit, with a pointer to the AnsiString *Text* term. This operation prepares each field in the design case record to accept a text input. This status for editing is posted to the Foreplane database table with the code *new_case_table->Post()*.

As mentioned above, there is an edit menu option for every field presented in the New Case window. Thus, it is possible for the user to edit fields individually with a reduced possibility of writing to the wrong edit field by mistake. The code that supports the menu options that facilitate the editing of individual fields is nearly identical to that illustrated for editing all the fields in a design case record at once. The only difference being that there is only one field under consideration. As an example, consider the code for editing the Component Description field.

```
void __fastcall Tnew_case::edit_comp_descripClick(TObject *Sender)
{
DBMemo1->ReadOnly=false;
new_case_table->Edit();
new_case_table->FieldByName("Component Description")->
Value=DBMemo1->Text;
new_case_table->Post();
}
```

When the Edit - Comp Description menu option is selected from the Edit menu the above function is called. It can be seen that the format of this function is identical to that for the function called by the Edit - All menu option as described above.

In addition to enabling the fields that support textual and numerical information to be edited the Edit menu also provides three menu options that allow the user to remove the graphics from any particular design case record. These menu options are Cut - Aircraft Type, Cut - Eng. Draw. and Cut - Structure Graphic. In terms of implementation the code that supports these menu options is identical. As such, consider the function below which is called when the Cut - Aircraft Type menu option is selected.

```
void __fastcall Tnew_case::cut_aircraft_typeClick(TObject *Sender)
{
        new_case_table->Edit();
        DBImage1->CutToClipboard();
        new_case_table->Post();
}
```

When this function is called it first puts the Foreplane database table in edit mode. Then the bitmap image residing in the Aircraft Type field is removed and placed in the Windows clipboard with the code *DBImage1->CutToClipboard()*. The term DBImage1 is the name by which the application recognises the VCL graphic component residing in the New Case window. The change to the Foreplane database table is posted with the code *new_case_table->Post()*.

### C.4.4. Delete

This section explains the operation of the code that facilitates the deletion of design case records residing in the case base. The code contained in the function called when the Delete Record option is selected from the Delete menu option in the New Case dialog box, first deletes the selected design case, then re-numbers the remaining records. The changes made are then posted to the Foreplane database table. The function supporting the Delete Record menu option is now given below.

```
void __fastcall Tnew_case::delete_recordClick(TObject *Sender)
{
new_case_table->Delete();

//re-numbering records
new_case_table->Edit();
new_case_table->FieldByName("Case Number")->AsInteger =1;

while(!new_case_table->Eof)
{
case_number=new_case_table->FieldByName("Case Number")->Value;
new_case_number=case_number+1;
new_case_table->Next();
new_case_table->Edit();
new_case_table->FieldByName("Case Number")->AsInteger = new_case_number;
}

while(!new_case_table->Bof)
{
new_case_table->Prior();
new_case_table->Edit();
}
new_case_table->Post();
}
```

The aspects of this function that relate to the re-numbering of design case records are fully documented in section C.4.2.1 which discusses the Insert menu option Insert Record. The reader who requires further information with respect to how design case records are re-numbered is referred to this section. The principle purpose of the above function is to enable the user to delete a record(s). This achieved with one line of code i.e., *new_case_table->Delete()*. This code removes a design case record from the Foreplane database table. Clearly, it could be suggested that it is unnecessary to present an entire function when only one line of code is of interest in context of

deleting a record. However, it is argued that it is important for the reader to appreciate the other coding requirements that are necessary as a result of this action.

### C.4.5. Scroll

This section presents the code that supports the four menu options in the Scroll menu option in the New Case window.

*Next*

```
void __fastcall Tnew_case::scroll_nextClick(TObject *Sender)
{
        new_case_table->Next();
}
```

*Previous*

```
void __fastcall Tnew_case::scroll_previousClick(TObject *Sender)
{
        new_case_table->Prior();
}
```

*First*

```
void __fastcall Tnew_case::scroll_to_firstClick(TObject *Sender)
{
        new_case_table->First();
}
```

*Last*

```
void __fastcall Tnew_case::scroll_to_lastClick(TObject *Sender)
{
        new_case_table->Last();
}
```

### C.4.6. Selection of the Best Case via the KBS

This section explains the operation of the code that facilitates the identification of the design case(s) in the case base that contain the best structure as defined by the ICES KBS.

Referring to section 9.2.2.4 of Chapter 9, which discusses the defining of the best structure in the KBS, it will be recalled that the seven section score totals relating to every structural type in both the structures and manufacturing rule base where added together. These combined seven section score totals for each structural type are then assigned to a series of arrays residing in the *structure* KBS_OUTPUT. This *structure* is declared in the header file *buffer.h*. The file *case_base_query.cpp* which supports the underlying operation of the Case Query and Jury window has access to the header file *buffer.h* i.e., it is included in the file's include list. As such, the file *case_base_query.cpp* has access to the seven section score totals placed in the arrays residing in the structure KBS_OUTPUT. Through access to the seven section score totals the file *case_base_query.cpp* is able to determine the best structure and make a comparison with the structural types declared within each of the design cases residing

in the case base. The first step necessary to make this comparison is to assign the seven section score totals corresponding to each structural type embraced by the KBS to the appropriate fields of the particular design cases i.e., cases which embrace the structural type in question. It should be noted that each design case record has seven fields declared whose purpose is to facilitate the assigning of the seven section score totals relating to the structural type that the particular design case record embraces. In the context of the user interface, these seven fields are not visible to the user. This is because the user does not have to interact with these fields and there display would serve no useful purpose.

To illustrate how the seven section score totals relating to each structural type are assigned to the seven fields of the appropriate design case consider the code example below which relates to the standard x-core structure. This example is taken from the function *rule_base_best_caseClick* which resides in the file *case_base_query.cpp* and is initiated when the user selects the Best Case option from the KBS menu.

```
if((new_case_table->FieldByName("Structure type")->
   AsString == "Standard X-core")&&
   (new_case_table->FieldByName("Adaptation Flag")->Value != 99))
{
AnsiString stand1,stand2,stand3,stand4,stand5,stand6,stand7;

stand1=struct_and_manu_output.standxcore_manu_struct_total1[0];
new_case_table->Edit();
new_case_table->FieldByName("Part Minimisation Seg 1")->Value = stand1;

stand2=struct_and_manu_output.standxcore_manu_struct_total2[0];
new_case_table->Edit();
new_case_table->FieldByName("Struct Char Seg 2")->Value = stand2;

stand3=struct_and_manu_output.standxcore_manu_struct_total3[0];
new_case_table->Edit();
new_case_table->FieldByName("Fastener Req Seg 3")->Value = stand3;

stand4=struct_and_manu_output.standxcore_manu_struct_total4[0];
new_case_table->Edit();
new_case_table->FieldByName("Fabrication Seg 4")->Value = stand4;

stand5=struct_and_manu_output.standxcore_manu_struct_total5[0];
new_case_table->Edit();
new_case_table->FieldByName("Environment Seg 5")->Value = stand5;

stand6=struct_and_manu_output.standxcore_manu_struct_total6[0];
new_case_table->Edit();
new_case_table->FieldByName("Geometric Seg 6")->Value = stand6;
stand7=struct_and_manu_output.standxcore_manu_struct_total7[0];
new_case_table->Edit();
new_case_table->FieldByName("Inter-connection Seg 7")->Value = stand7;
}
```

Before discussing the above code in detail it should be pointed out that similar code exists for every structural type embraced by the ICES prototype. In addition, the above code and that corresponding to other structures is embraced within a while loop i.e., *while(!new_case_table->Eof)*. This while loop ensures that every design case record in the case base is considered by the system.

Turning attention now to the code example above and taking the code in the order of its declaration. The code starts with an *if* conditional statement. This statement first compares the string value assigned to the design case record field *Structure Type* with the string "Standard X-core". The *if* conditional statement then compares the numerical value assigned to the *Adaptation Flag* field with the value 99. If the string assigned to the *Structure Type* field does correspond to the string "Standard X-core" and the numerical value assigned to the field *Adaptation Flag* does not equal 99 then the *if* conditional statement may be entered.

The code embraced by the two braces of the *if* conditional statement starts by declaring seven string objects instantiated from the AnsiString class i.e., *stand1*, *stand2*, *stand3*, *stand4*, *stand5*, *stand6*, and *stand7*. Having declared these seven string objects, the code then takes the value residing in the array *standxcore_manu_struct_total1* which corresponds to the first of the seven sections relating to standard x-core declared in the *structure* KBS_OUTPUT and assigns it to the string object *stand1*. The Foreplane database table is then put into edit mode. The numerical value assigned to stand1 is then placed in the field *Part Minimisation Seg1* of the particular design case record in question. As illustrated in the code example above, this process is repeated a further six times so that all the values assigned to the arrays corresponding to standard x-core declared in the *structure* KBS_OUTPUT are placed in the appropriate fields of the design case record i.e. the fields corresponding to the seven section score totals.

With the seven section score totals assigned to the appropriate fields of all the design case records residing in the case base it is then necessary to identify the case whose seven section score totals summed is the highest. This case will then be identified as that case which possess the best structure as defined by the ICES KBS. The code that facilitates this operation is given below; it is initiated within the function *rule_base_best_caseClick* which as indicated previously is declared in the file *case_base_query.cpp*

```
new_case_table->First();

while(!new_case_table->Eof)
{
best_case_number=new_case_table->FieldByName("Case Number")->Value;
segment1=new_case_table->FieldByName("Part Minimisation Seg1")->Value;
segment2=new_case_table->FieldByName("Struct Char Seg 2")->Value;
segment3=new_case_table->FieldByName("Fastener Req Seg3")->Value;
segment4=new_case_table->FieldByName("Fabrication Seg4")->Value;
segment5=new_case_table->FieldByName("Environment Seg5")->Value;
segment6=new_case_table->FieldByName("Geometric Seg6")->Value;
```

```
segment7=new_case_table->FieldByName("Inter-connection Seg7")->Value;
segment_total = (segment1+segment2+segment3+segment4+segment5+
                 (segment6+segment7);
if(segment_total>=best_case_kbs)
  {
  best_case_kbs=segment_total;
  best_case_selection_number = best_case_number;
  }
new_case_table->Next();
}
```

Taking the above code in the sequence that it is declared. The code *new_case_table->First()* takes the application to the first design case record in the case base. The code then declares a *while* loop i.e., *while(!new_case_table->Eof)*, which says while the Foreplane database table is not at the end of the table continue. Inside the *while* loop, the identification number of the first design case record is assigned to the integer variable *best_case_number* with the code *best_case_number=new_case_table->FieldByName("Case Number")->Value.* The seven lines of code that follow this operation then take the numerical values residing in the seven fields in the first design case record relating to the seven section score totals and assign them to seven float variables i.e., *segment1* to *segment7.* The values assigned to these float variables are then summed and assigned to the single float variable *segment_total.*

The code now declares an *if* conditional statement within the afore mentioned while loop i.e., *if(segment_total>=best_case_kbs).* The test in the *if* conditional statement tests to see if the value assigned to the float variable *segment_total* is equal or greater than the float variable *best_case_kbs.* It should be noted that the default value assigned to the float variable *best_case_kbs* is 0. As such, when the *if* conditional statement is first tested the float variable *segment_total* will always be greater than the float variable *best_case_kbs.* Inside the braces supported by the *if* conditional statement the float variable *segment_total* is assigned to the float variable *best_case_kbs.* In addition, the integer variable *best_case_number* is assigned to the integer variable *best_case_selection_number.* This inter-changing of integer variables has the effect of assigning the design case record identification number to a successful test of the *if* conditional statement i.e., when the float variable *segment_total* corresponding to a design case record is greater or equal to the float variable *best_case_kbs* then the design case record identification number is temporally retained by the integer variable *best_case_selection_number.*

Following the *if* conditional statement the code *new_case_table->Next()* takes the application to the next record in the case base. The above *while* loop iterates through for all the design case records in the case base and exits when the end of the Foreplane database table is reached. At this juncture, the integer variable *best_case_selection_number* will hold the identification number for the design case record which has the highest value for the summation of all the seven section score totals i.e., this case contains the best structure as defined by the ICES KBS.

Having identified the design case within the case base that contains the best structure it is now necessary for this case to be displayed to the user. This is achieved with the following code residing in the function *rule_base_best_caseClick.*

```
new_case_table->First();

do
{
case_number_value=new_case_table->FieldByName("Case Number")->Value;
        if((Bookmark == NULL)&&(case_number == best_case_selection_number))
        {
        Bookmark=new_case_table->GetBookmark();
        }

new_case_table->Next();
}while(!new_case_table->Eof);

if(Bookmark != NULL)
{
        new_case_table->GotoBookmark(Bookmark);
        new_case_table->FreeBookmark(Bookmark);
        Bookmark = NULL;
}
```

The above code first takes the application to the first design case record in the case base. The code then declares a *do while* loop. Inside the *do while* loop the identification number of the first design case record is assigned to the integer variable *case_number_value.* The code continues by declaring the *if* conditional statement *if((Bookmark == NULL)&&(case_number == best_case_selection_number))*. Before discussing the meaning of this *if* conditional statement it is first necessary to introduce bookmarks in the context of C++ Builder. The database capability residing in C++ Builder enables the user to bookmark any desired record i.e., mark the record as being important. Having bookmarked a record it then possible for the user to return to this record as and when desired within the operation of the application. The C++ Builder development environment also permits bookmarks assigned to records to be removed.

Returning now to the *if* conditional statement, the code *(Bookmark == NULL)* tests to see if the first design case record has a bookmark assigned to it. The second part of the *if* conditional statement tests to see if the first design case record identification number i.e., *case_number_value*, is the same as the integer value assigned to the integer variable *best_case_selection_number*. As indicated above, the value held by the *best_case_selection_number* corresponds to the design case record which has the highest value for the summation of all the seven section score totals and as such contains the best structure as defined by the KBS. If there is no bookmark assigned to the first design case record and the integer variables *case_number* and *best_case_selection_number* are equal the *if* conditional statement may be entered. The code inside the if conditional statement assigns a bookmark to the first design case record i.e., *Bookmark=new_case_table->GetBookmark()*. The *GetBookmark* call

returns a variable of type pointer, which in fact is just a pointer to a Bookmark. A Bookmark pointer contains enough information to enable C++ Builder to find the location to which it refers.

If either of the tests in the *if* conditional statement mentioned above fail then the code inside the *if* conditional statement cannot be accessed. The application then moves on to the next record i.e., *new_case_table->Next()*, and the above process is repeated. This continues until the test in the *while* part of the *do while* loop fails i.e., the application reaches the end of the Foreplane database table and there are no more cases in the case base to be tested. When the *do while* loop exits the application goes back to the first design case record.

Clearly, once a bookmark has been assigned to the design case record which contains the best structure as defined by the KBS it is desirable that this record be displayed by the application. In addition, it also necessary that the bookmark be removed from the particular design case record once it has been displayed so that the application can be run again under a different set of conditions. The code that supports these operations is given below.

```
if(Bookmark != NULL)
{
        new_case_table->GotoBookmark(Bookmark);
        new_case_table->FreeBookmark(Bookmark);
        Bookmark = NULL;
}
```

This code follows on from the previously mentioned *do while* loop. The *do while* loop firstly cycles through all the design case records in the case base and assigns a bookmark to the appropriate case. When the application exits the *do while* loop it goes back to the first design case record in the case base and then again cycles through all the cases in the case base applying the *if* conditional test above i.e., *if(Bookmark != NULL)*. This *if* conditional statement tests to see if each case in the case base has been assigned a bookmark. If it is found that a design case record has been assigned a bookmark then the *if* conditional statement is entered and the code *new_case_table->GotoBookmark(Bookmark)* takes the application to the appropriate design case record and displays it in the Case Query and Jury window. The next two lines of code free the memory assigned for the bookmark and assign a NULL value to the pointer *Bookmark*.

### C.4.7. Selection of the Best Case via Nearest Neighbour Matching and Applying the Jury Technique

This section explains the operation of the code that supports the selection of the best case through the application of nearest neighbour matching and the application of the jury technique. As indicated in section 9.2.3.2 of Chapter 9, the Case Query and Jury window has a series of edit boxes listed down its left-hand size. These edit boxes permit the user to enter particular known performance specifications, see Figure 9.24. Once specifications have been entered into these edit boxes the user should select the Best Case menu option from the Jury menu. The Best Case menu option calls the

function *best_case_runClick* which resides within the file *case_base_query.cpp* which supports the operation of the Case Query and Jury window. The operation of the Best Case menu option has the effect of deriving the best case through nearest neighbour matching. As part of the application of the nearest neighbour matching process the file *case_base_query.cpp* first initialises a series of float variables as follows.

```
//importance ratings
very_important = 1;
important = 0.8;
moderate = 0.4,
low_importance = 0.2;
no_importance = 0;
total_importance = (very_important+important+moderate+low_importance+
                    no_importance);
//Degree of match
match = 1;
close_match = 0.8;
moderate_match = 0.4;
poor_match = 0.2;
no_match = 0;
```

The importance float variables are an indication of how important any particular specification is in the context of the design process. The match float variables are a measure of how closely a specification attributed to a design case matches the specifications entered by the user. The role of both of these sets of variables will become apparent as the implementation of the nearest neighbour matching technique is explained.

As indicated above, the selection of the Best Case menu option from the Jury menu has the effect of calling the function *best_case_runClick*. This function first takes the application to the first design case record in the case base. Then it takes the specifications entered by the user and the specifications attributed to the first design case record, including the design case identification number, and assigns them to a series of float variables. The code that supports these operations is given below.

```
//Go to first record in case base
new_case_table->First();
//get_case_number
best_case=new_case_table->FieldByName("Case Number")->Value;
//input specifications
probe_max_shear = get_number(Edit3);
probe_min_shear = get_number(Edit4);
probe_max_bending = get_number(Edit5);
probe_min_bending = get_number(Edit6);
probe_max_torque = get_number(Edit7);
probe_min_torque = get_number(Edit8);
probe_tensile_yield_stress = get_number(Edit9);
probe_uts = get_number(Edit10);
probe_youngs_modulus = get_number(Edit11);
```

```
probe_density = get_number(Edit12);
//case specifications
db_max_shear=new_case_table->
FieldByName("Max Pos Shear Load kN")->Value;
db_min_shear=new_case_table->
FieldByName("Max Neg Shear Load kN")->Value;
db_max_bending=new_case_table->
FieldByName("Max Pos Bend Load kN")->Value;
db_min_bending=new_case_table->
FieldByName("Max Neg Bend Load kN")->Value;
```

The operation of all the above code has been explained previously in Appendix C. As such, the reader is directed to sections C.2.3.2 and C.4.6 for a full explanation of the code functionality. With the specifications entered by the user and the specifications attributed to the first design case record assigned to a series of float variables, it is now possible to determine the nearest neighbour weighting for every individual specification. Taking the maximum positive shear load specification as an example, consider the following code.

```
if(probe_max_shear !=0)
{
max_shear_value=db_max_shear/probe_max_shear;
        if(max_shear_value>=1)
        {
        max_shear_match=match;
        max_shear_weighting=max_shear_match*very_important;
        }
        if((max_shear_value>0.9)&&(max_shear_value<1))
        {
        max_shear_match=close_match;
        max_shear_weighting=max_shear_match*very_important;
        }
        if((max_shear_value>0.8)&&(max_shear_value<=0.9))
        {
        max_shear_match=moderate_match;
        max_shear_weighting=max_shear_match*very_important;
        }
        if((max_shear_value>0.5)&&(max_shear_value<=0.8))
        {
        max_shear_match=poor_match;
        max_shear_weighting=max_shear_match*very_important;
        }
        if(max_shear_value<=0.5)
        {
        max_shear_match=no_match;
        max_shear_weighting=max_shear_match*very_important;
        }
}
```

The above code is encapsulated within a single *if* conditional statement i.e., *if(probe_max_shear !=0)*. This *if* conditional statement performs a test to check that a numerical value has been entered by the user in the edit box corresponding to maximum positive shear load. That is, the *if* conditional statement checks to see that the numerical value assigned to the float variable *probe_max_shear* does not equal zero. Providing that *probe_max_shear* does not equal zero the code residing in the *if* conditional statement braces may be accessed.

It is assumed that there is a numerical value assigned to the float variable *probe_max_shear*. Thus, the first line of code in *if* conditional braces takes the value assigned to the variable corresponding maximum positive shear load in the first design case record i.e., *db_max_shear*, and divides it by the value assigned to the variable *probe_max_shear*. The result of this operation is assigned to the float variable *max_shear_value*. It can be seen that the code that follows this operation is a series of five *if* conditional statements. These *if* conditional statements test the numerical value assigned to the float variable *max_shear_value*. If the value falls in the range of any of the *if* conditional statements then the code relating to that particular *if* conditional statement may be accessed. For example, if the value assigned to *max_shear_value* where to equal 0.95 then the following *if* conditional statement code may be accessed.

```
if((max_shear_value>0.9)&&(max_shear_value<1))
{
max_shear_match=close_match;
max_shear_weighting=max_shear_match*very_important;
}
```

Once one of the five *if* conditional statements is accessed, the appropriate variable corresponding to the quality of the match between the variables *db_max_shear* and *probe_max_shear* as assigned to the variable *max_shear_value*, is assigned to the float variable *max_shear_match*. The float variable *max_shear_match* is then multiplied by the float variable corresponding to the degree of importance of the particular specification. In this instance the specification is very important and *max_shear_match* is multiplied by the float variable *very_important*. The result is then assigned to the float variable *max_shear_weighting*.

The procedure described above is repeated for every specification entered by the user and applied to all the design case records residing in the case base. A weighted value is assigned to an equivalent float variable corresponding to the variable *max_shear_weighting* as described above for each specification in each design case. For each design case the weighted float variables are added together and then divided by the float variable *total_importance*. The numerical value obtained as a result of this operation is assigned to the float variable *near_neighbour* as illustrated below.

```
near_neighbour = (max_shear_weighting+min_shear_weighting+
                  max_bending_weighting+min_bending_weighting+
                  max_torque_weighting+min_torque_weighting+
                 tensile_yield_stress_weighting+uts_weighting+
                 youngs_modulus_weighting+density_weighting)/total_importance;
```

Having derived a value for the float variable *near_neighbour* for each design case residing in the case base it possible to identify the case that most closely matches the user's input specifications. The nearest neighbour matching technique continues by assigning the value assigned to the first design case *near_neighbour* float variable to a temporary static float variable i.e., *static temp = near_neighbour*. The application then compares all the values assigned to the *near_neighbour* variable for all the remaining design cases with the temporary float variable *temp*. If the value assigned to *near_neighbour* by another design case is found to be equal or greater than the value assigned to the variable *temp* then this value becomes the new value assigned to the temporary variable *temp*. The code that supports this operation is given below.

```
if((near_neighbour>=temp)&&
 (new_case_table->FieldByName("Flag")->Value !=99)&&
 (new_case_table->FieldByName("Reject Flag")->Value !=99)&&
 (new_case_table->FieldByName("Adaptation Flag")-> !=99))
 {
 best_case=best_case_number;
 temp=near_neighbour;
 }
```

It can be seen that the above code has an extensive *if* conditional test statement. The code within this statement in addition to the nearest neighbour technique, relates to the operation of the jury technique and case adaptation. The *if* conditional statement presents four test conditions that must be fulfilled in order for the code within the braces to be accessed. The first condition tests to see if the *near_neighbour* float variable is equal to or greater than the float variable *temp*. The second condition tests to see if the numerical value 99 is assigned to the design case record field *Flag*. This is a test to see if the case has been identified as a best case before. The third condition tests to see if the numerical value 99 has been assigned to the design case record field *Reject Flag*. This is a test to see whether the case has been rejected previously as the best case. The final condition tests to see if the numerical value 99 has been assigned to the design case record field *Adaptation Flag*. This is a test to see whether the case is an adapted case or not. If the *near_neighbour* variable is equal to or greater than the variable *temp* and the numerical value 99 has not been assigned to the *Flag*, *Reject Flag* and *Adaptation Flag* fields the code within the *if* conditional statement braces may be accessed. The underlying functionality of this code is explained in section C.4.6, which discusses the selection of the best case via the KBS. However, in order to continue this discussion, it is should be appreciated that the purpose of this code is to temporally retain the design case record identification number which corresponds to the case which represents the closest match with the user input specifications i.e., nearest neighbour.

It is now necessary to display the best case derived as a result of the nearest neighbour matching technique. The code residing in the function *best_case_runClick* that supports this operation is given below.

```
new_case_table->First();
do
{
case_number_value=new_case_table->FieldByName("Case Number")->Value;
```

```
if((Bookmark == NULL)&&(case_number_value==best_case))
{
//assign a bookmark
Bookmark=new_case_table->GetBookmark();
//assign a flag
new_case_table->Edit();
new_case_table->FieldByName("Flag")->AsInteger = 99;
temp = 0;
}
new_case_table->Next();
}while (!new_case_table->Eof);

if(Bookmark != NULL)
{
        new_case_table->GotoBookmark(Bookmark);
        new_case_table->FreeBookmark(Bookmark);
        Bookmark = NULL;
        temp = 0;
}
```

The above code first takes the application to the first design case record in the case base. The code then declares a *do while* loop. Inside the *do while* loop the identification number of the first design case is assigned to the integer variable *case_number_value*. The code then declares an *if* conditional statement which tests to see if a bookmark is already applied to the first design case record. In addition, the *if* conditional statement tests to see if the numerical value assigned to the *case_number_value* is equal to the integer variable *best_case*. If both these conditions of the *if* conditional statement are satisfied the code residing within the braces of the *if* conditional statement may be accessed.

Assuming the conditions of the *if* conditional statement are met the first design case is assigned a bookmark i.e., *Bookmark=new_case_table->GetBookmark()*. Following the assigning of the bookmark the numerical value 99 is assigned to the first design case field *Flag* i.e., *new_case_table->FieldByName("Flag")->AsInteger = 99*. This value assigned to the field *Flag* tells the system that this particular case in the case base is the best case. Having identified the best case the application then returns the temporary float variable *temp* back to zero i.e., *temp = 0*. It should be noted that if the *if* conditional statement conditions where not met the application would not access the *if* conditional statement but rather cycle round to the next design case in the case base i.e., *new_case_table->Next()*, and the process would be repeated. This process would continue until the application reached the end of the Foreplane database table, at which point, the application would exit the *do while* loop.

Having bookmarked the best case as defined by the nearest neighbour matching process it is now necessary to display it. This is achieved through the *if* conditional statement code following the *do while* loop i.e., *if(Bookmark != NULL)*. Each design case is tested against the *if* conditional statement test. If a case is found to have a bookmark assigned to it the code contained in the *if* conditional statement may be

accessed. The code within the braces of the *if* conditional statement first takes the application to the bookmarked design case record and displays it in the Case Query and Jury window. The code then frees the memory allocated for the bookmark and dismisses the bookmark by assigning a *NULL* value to it.

The code that facilities the displaying the next best design case is almost identical to the code that which is utilised to determine the best case itself. The differences will now be discussed. It will be recalled from the above explanation that the code that supports the derivation of the best design case assigns the numerical value 99 to the design case field *Flag* i.e., *new_case_table->FieldByName("Flag")->AsInteger = 99.* When the nearest neighbour operation is performed to determine the next best case the assigning of this value of 99 to the field *Flag* assures that the previously determined best case is not included in the derivation of the next best case i.e., the numerical value of 99 assigned to the *Flag* field excludes the best case. The next best design case record is bookmarked in an identical method as described for deriving the best case. The next best case is then displayed in Case Query and Jury window in a similar fashion to method used to display the best case. In addition to freeing the memory allocated for the bookmark corresponding to the next best case and assigning a NULL value to the bookmark pointer itself, the system cancels all the numerical values of 99 assigned to the *Flag* fields i.e., for both the best case and the next best case. The code that supports this operation is given below.

```
new_case_table->First();
do
{
new_case_table->Edit();
new_case_table->FieldByName("Flag")->AsInteger = 0;
new_case_table->Next();
}while(!new_case_table->Eof);
```

This code takes the application to the first design case in the case base. A *do while* loop is then declared. The code within this *do while* loop then places the Foreplane database table in edit mode i.e., *new_case_table->Edit()*. Following this operation, zero is assigned to the design case field Flag i.e., *new_case_table->FieldByName("Flag")->AsInteger = 0*. The application then moves on to the next case in the case base. This procedure is continually repeated until the end of the Foreplane database table is reached, at which point the *do while* loop exits. When the *do while* loop exits all the *Flag* fields corresponding to all the design case records in the case base will be assigned the numerical value 0 i.e., all the 99 numerical 'flags' assigned to the Flag fields are cancelled.

As the 99 numerical flag has been removed from all the design case Flag fields it is possible for the user to re-examine the best case by selecting the Best Case menu option from the Jury menu. Thus, the user can move the application to and from the best case to the next best case as desired. To reject the best case the user should select the Reject Case option from the Jury menu. When the user selects the Reject Case menu option the following function *reject_caseClick* is called.

```
void __fastcall Tspecifications_abstraction::reject_caseClick(TObject *Sender)
{
        new_case_table->Edit();
        new_case_table->FieldByName("Reject Flag")->AsInteger = 99;
}
```

This function first places the Foreplane database table in edit mode. It then assigns the numerical value 99 to the best design case record field *Reject Flag*. The effect of assigning the 99 numerical 'flag' to the field *Reject Flag* is to mark the current best case as rejected. Once rejected this case can no longer be considered by the system or be displayed in the Case Query and Jury window.

When the user has finished querying the case base and decides to exit the Case Query and Jury window it is necessary to remove all the numerical 99 'flags' and bookmarks assigned to design cases residing in the case base so that the case base is ready to be queried afresh the next time the application is used. When the user selects the Close menu option the following function is called.

```
void __fastcall Tspecifications_abstraction::new_case_exitClick(TObject *Sender)
{
new_case_table->First();

//remove all flags
        do
        {
new_case_table->Edit();
new_case_table->FieldByName("Flag")->AsInteger = 0;
new_case_table->Edit();
new_case_table->FieldByName("Reject Flag")->AsInteger = 0;
new_case_table->Next();
}while(!new_case_table->Eof);

if(Bookmark != NULL)
{
new_case_table->FreeBookmark(Bookmark);
Bookmark = NULL;
}
        new_case_table->First();
        new_case_table->Close();

        Close();
}
```

It is not proposed to discuss the above code in detail as all aspects of the above function have covered previously in this appendix. However, taking the function in the order of its declaration. It first takes the application to the first record in the case base. Following this, a *do while* loop is declared, in which zero is assigned to the design case record fields *Flag* and *Reject Flag* for every design case in the case base.

The function then checks that no bookmarks are assigned to any design cases. If they are, the memory associated with any bookmark is freed and the bookmark itself is assigned the value NULL. Having removed any numerical 'flags' and bookmarks the application returns to the first design case record in the case base. The application then closes down the Case Query and Jury window.

### C.4.8. Case Adaptation

This section explains the code that supports the case adaptation facility provided in the ICES case base as discussed in section 9.2.3.2.3 of Chapter 9. When the Create Adaptation Platform menu option is selected from the Adaptation menu in the Case Query and Jury window the function *adaptation_platformClick* is called. The code supported within this function apart from one important feature is identical to the code documented in section C.4.2.1 which discusses the insertion of a new design case record. The reader who wishes to appreciate the underlying code supporting the Create Adaptation Platform is directed to this appendix. The one aspect of the code supported within the function *adaptation_platformClick* that is different than that employed to create a new case record is as follows.

```
new_case_table->FieldByName("Adaptation Flag")->AsInteger = 99
```

This line of code assigns the numerical value 99 to the new case record field *Adaptation Flag*. The purpose of assigning this 'flag' to the new case record is to mark it as an adapted case. As discussed in section 7.3.4, there is a need to identify adapted cases as being different from real aircraft cases. This is because they are not as creditable as fully validated cases e.g., the European Fighter Aircraft (EFA) or the Experimental Aircraft Programme (EAP).

With the blank platform created on to which to build the adapted case the user should copy an existing real case onto this platform. The user should either select the best design case record or scroll through the cases residing in the case base until the desired one is displayed. With the desired design case record displayed the user should select the menu option Bookmark Case from the Adaptation menu option. The Bookmark Case menu option calls the following function which assigns a bookmark to the selected design case record.

```
void __fastcall Tspecifications_abstraction::case_bookmarkClick(TObject *Sender)
{
        if(Bookmark == NULL)
        {
        Bookmark=new_case_table->GetBookmark();
        }
}
```

The above code is relatively straight forward. The *if* conditional statement checks to see if a bookmark is already assigned to the design case record. If there is not a bookmark already assigned to the case then one is assigned.

With the desired case selected and a bookmark assigned, the user should now select the Copy Record menu option from the Adaptation menu. This menu option will cause the selected case to be copied to the blank adaptation platform. The code that supports this copying operation is contained in the function *copy_recordClick*. A very abridged version of this function is presented below.

```
void __fastcall Tspecifications_abstraction::copy_recordClick(TObject *Sender)
{
        //Graphics
        if(Bookmark != NULL)
        {
new_case_table->GotoBookmark(Bookmark);
DBImage1->CopyToClipboard();
new_case_table->First();
DBImage1->PasteFromClipboard();
}

//Memo box
if(Bookmark != NULL)
{
new_case_table->GotoBookmark(Bookmark);
int Size = DBMemo1->GetTextLen();
char *name = new char[++Size];
DBMemo1->GetTextBuf(name, Size);
new_case_table->First();
new_case_table->Edit();
DBMemo1->Text = name;
new_case_table->Post();
delete name;
}

//Edit box
if(Bookmark !=NULL)
{
new_case_table->GotoBookmark(Bookmark);
int Size = DBEdit11->GetTextLen();
char *name = new char[++Size];
DBEdit11->GetTextBuf(name, Size);
        new_case_table->First();
        new_case_table->Edit();
        DBEdit11->Text = name;
        new_case_table->Post();
        delete name;
        }
}
```

The code presented in the abridged version of the function *copy_recordClick* illustrates how the three types of information represented in the selected design case record can be copied onto the blank adaptation platform i.e., that information held in graphics, memo and edit boxes. It can be seen that the above function contains three *if*

conditional statements. Each of these *if* conditional statements tests to see if the design case record selected has been bookmarked. If the case has been bookmarked then the code contained within the braces of the each of the three *if* conditional statements may be accessed.

Clearly, in the context of providing an explanation of the above code it is assumed that the selected case is bookmarked. Taking the code in each of the *if* conditional statements in order of declaration. The code in the first *if* conditional statement provides the capability to copy graphics from one design case record to another. The first line of code i.e., *new_case_table->GotoBookmark(Bookmark)*, takes the application to the bookmarked case. The image residing in the field corresponding to the VCL component *DBImage1* is copied to the Windows clipboard i.e., *DBImage1->CopyToClipboard()*. The code *new_case_table->First()* then takes the application to the first design case record in the case base. It should be noted that a blank adaptation platform is always created in the first position in the case base. The code *DBImage1->PasteFromClipboard()* then takes the image from the Windows clipboard and inserts it in the VCL component *DBImage1* corresponding to appropriate graphics field in the blank adaptation platform.

Turning attention now to the next two *if* conditional statements i.e., embracing the copying of information held in memo and edit boxes. The code contained in these *if* conditional statements is nearly identical. It is proposed to discuss the copying of information from the memo box field components to the adaptation platform and then highlight the differences in the context of information contained in edit boxes. The first line of code in the second *if* conditional statement braces again takes the application to the bookmarked case. The length of the text string residing in the memo box *DBMemo1* is determined with the code *int Size = DBMemo1->GetTextLen()*. Space in memory is then allocated for this string with the code *char *name = new char[++Size]*. The text contained in the memo box is then placed in the buffer *name* i.e., *DBMemo1->GetTextBuf(name, Size)*. Having placed the text in the buffer, the code takes the application to the first design case record in the application i.e., the adaptation platform, and puts the Foreplane database table in edit mode. The text held in the buffer *name* is then placed in the *DBMemo1* VCL component corresponding to the appropriate text field in the adaptation platform i.e., *DBMemo1->Text = name*. The code finally posts changes made to the case base to the Foreplane database table and frees the memory allocated to the buffer *name*.

In the context of the third *if* conditional statement, which facilitates the copying of information contained in edit boxes to the adaptation platform. The only difference with respect to the code used to copy information contained in memo boxes to the adaptation platform as described above, is the declaration of the VCL component type name i.e., *DBEdit11* rather than *DBMemo1*.

With an existing real case copied onto the adaptation platform the user should now select the Free Case Bookmark from the Adaptation menu. This menu option calls the function *adaptation_free_bookmarkClick* which frees the memory allocated to the bookmark and assigns the *NULL* value to the bookmark pointer itself. Having freed the bookmark, the user should scroll the application to the desired design case record which contains the aspect which it is intended to substitute in the equivalent

field in case presented in the adaptation platform. With the appropriate case record displayed, the user should again select the Bookmark Case menu option from the Adaptation menu. Following the bookmarking of the selected case record the user should select the Field Adaptation menu option from the Adaptation menu. The Field Adaptation menu provides access to a further sub-menu. This sub-menu enables the user to select the desired field of the displayed design case that it is required to copy to the adaptation platform. In terms of operation, the code that supports this copying of the information in the selected field to the adaptation platform is identical to that discussed above in the context of copying an entire case to the adaptation platform. After the appropriate sub-menu option(s) from the Field Adaptation menu option have been selected the user may scroll back to the adaptation platform where the field(s) selected by the user can be seen to have been substituted into the appropriate fields of the adapted case.

As indicated earlier in this section, adapted cases are assigned a numerical 99 'flag' value in the field *Adaptation Flag*. The purpose of this flag is to keep real cases separate from adapted cases. By default, adapted cases are not included in queries of the case base. However, if the user wishes to query all cases in the case base the Adaptation menu provides the menu option Include Adapted Cases. This menu option removes the numerical 99 flag and all design cases in the case base are treated the same. The user may also re-apply the adaptation flag by selecting the Apply Adaptation Flag menu option from the Adaptation menu.

## C.5. Expansion of ICES Prototype Code

This section outlines the principle areas where the existing ICES prototype may be expanded. It is assumed that the developer has access to the ICES prototype source code and Borland C++ Builder Professional.

### C.5.1. KBS - Entering structures into the Structure Identification dialog box.

As indicated in section C.2.2, when the user enters the component it is desired to design in the Structure Identification dialog box this has the effect of enabling menu options residing beneath the KBS main menu option. The code that supports this capability is given below:

```
void __fastcall Tstruct_ident::BitBtn1Click(TObject *Sender)
{
AnsiString s;
int Size = Edit1->GetTextLen(); //get length of string in Edit1
char *name = new char[++Size]; //allocate space for string
Edit1->GetTextBuf(name,Size); //puts Edit1 text into buffer
s=name;
delete name; //frees memory allocated to buffer

if(s == "foreplane")
{
aero_struct.foreplane[0] =1;
}
```

```
}
```

The detailed explanation of the above code is provided in section C.2.2. At present the above code limits the type of structure that may be entered into the Structure Identification dialog box to a foreplane. That is, the *if* statement *if(s == "foreplane")* compares the string object *s* to the string *foreplane*. If *s* equals the *foreplane* string then the value 1 is assigned to a global variable residing in a structure array in the header file *buffer.h*.

To expand the operation of the Structure Identification dialog box so that it may facilitate the entry of other structural types into the system is a two step process. Firstly, the above code must be expanded so that the string object s may be compared with other possible aircraft structural components. Secondly, a further global variables must be created, each of which correspond to each of the additional aircraft components. The code below illustrates how this is achieved in the context of an aircraft wing:

```
void __fastcall Tstruct_ident::BitBtn1Click(TObject *Sender)
{
        AnsiString s;
        int Size = Edit1->GetTextLen(); //get length of string in Edit1
        char *name = new char[++Size]; //allocate space for string
        Edit1->GetTextBuf(name,Size); //puts Edit1 text into buffer
        s=name;
        delete name; //frees memory allocated to buffer

        if(s == "foreplane")
        {
        aero_struct.foreplane[0] =1;
        }

        if(s == "wing")
        {
        aero_struct.wing[0] =1;
        }
}
```

The above code is expanded to permit the string object s to be compared with the string "wing". If the two are equal the value 1 is assign to the global variable array aero_struct.wing[0]. This variable 1 may now be made visible to all other components within the ICES prototype need 'see' it.

```
typedef struct
{
int foreplane[50];
int wing[50];
}AIRCRAFT_STRUCTURE;
```

The *structure AIRCRAFT_STRUCTURE* above is taken from the header file *buffer.h*. The code example expands the *structure* to support the global variable integer array *int wing[50]*.

### C.5.2. Adding new menu options to the ICES prototype application

This section explains how new menu options may be added to the ICES prototype. Figure C.1 shows the ICES prototype development form. This is nearly identical to the application start-up window. The only visible difference is that a main menu icon is present in the left-hand top corner of the development form. Clicking on this icon enables the developer to edit the main menu i.e., add or delete menu items or edit existing ones. When the developer clicks on the main menu icon the main menu edit form will appear as indicated in Figure C.2. The developer can move through the main menu as he or she would a normal application. When the developer arrives at a point in the menu where it is desired to insert a new menu option the right mouse button should be clicked; this then presents the user with a menu providing a range of menu edit options as indicated in Figure C.3. The developer should now select the insert menu option; a blank menu item will now be inserted into the application. The developer should now write the name to be displayed on the newly inserted menu option in the Object Inspector in the Caption field provided under the component Properties list, see Figure D.3. The developer should also give the menu option a Name; this is further field in Object Inspector component property list. The Name property is the name by which the ICES prototype application will recognise the new menu option. The main menu edit form can now be closed down. The C++ Builder application will automatically update the main menu.

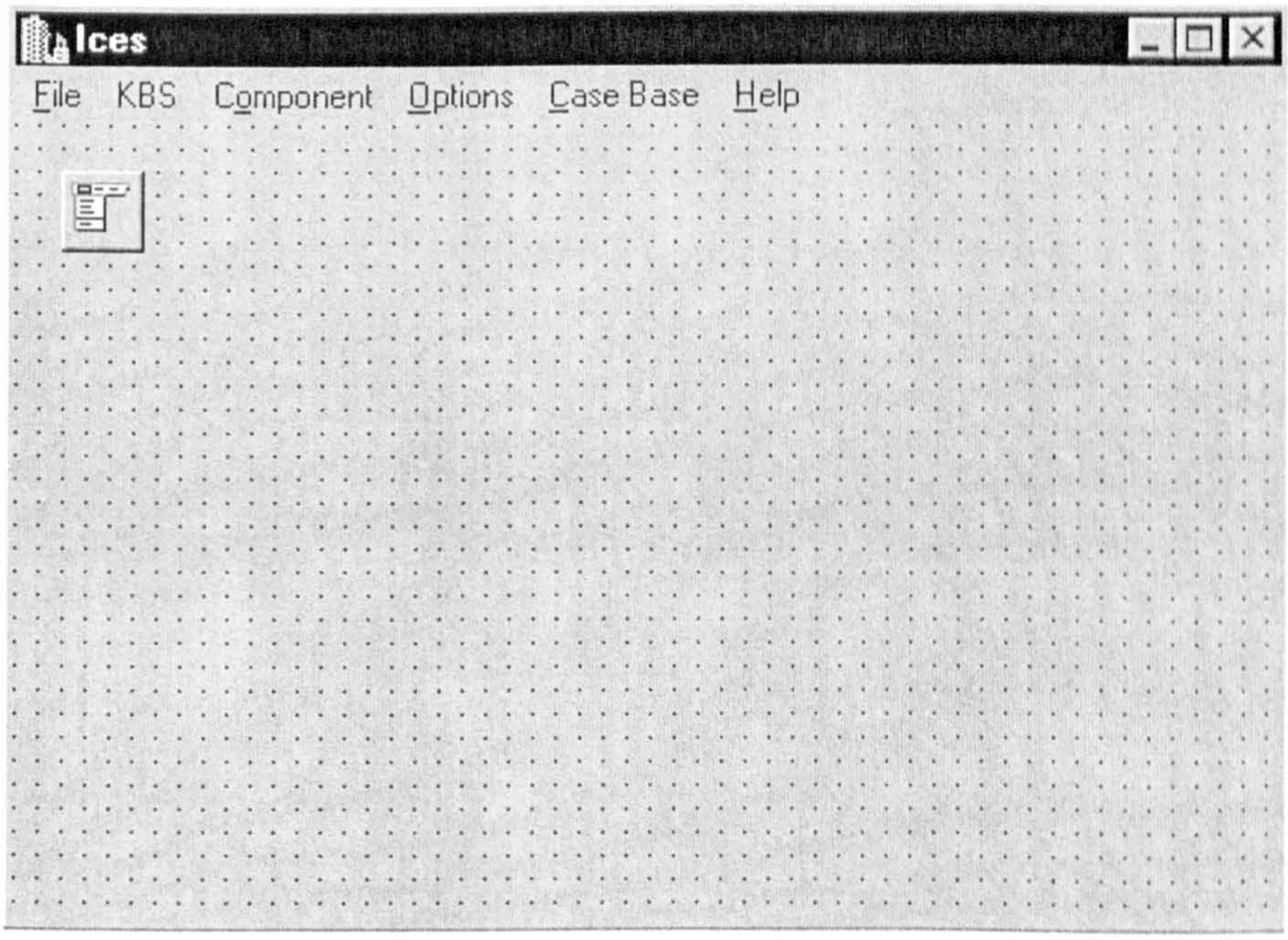



Figure C.1, The ICES Prototype Development Form

### C.5.3. KBS - Enabling additional questions to be asked in the Questions dialog box

The purpose of the Questions dialog box as indicated in section 9.2.1.3.2 of Chapter 9 is to obtain further details with respect to the component it is proposed to design i.e.,

interrogate the user. To add additional questions to those already present in the Questions dialog box the procedure below should be followed.

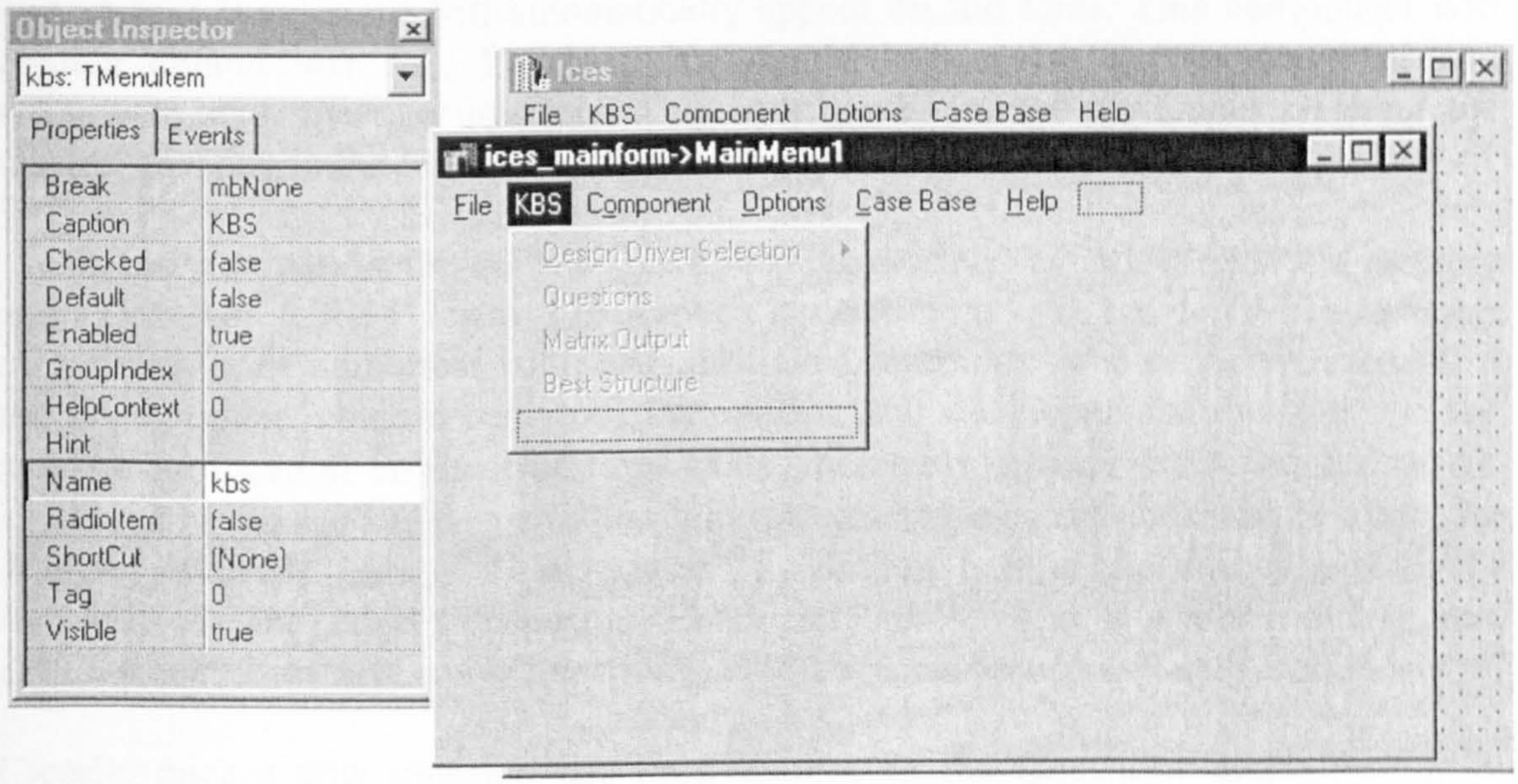



Figure C.2, The Main Menu Edit Form

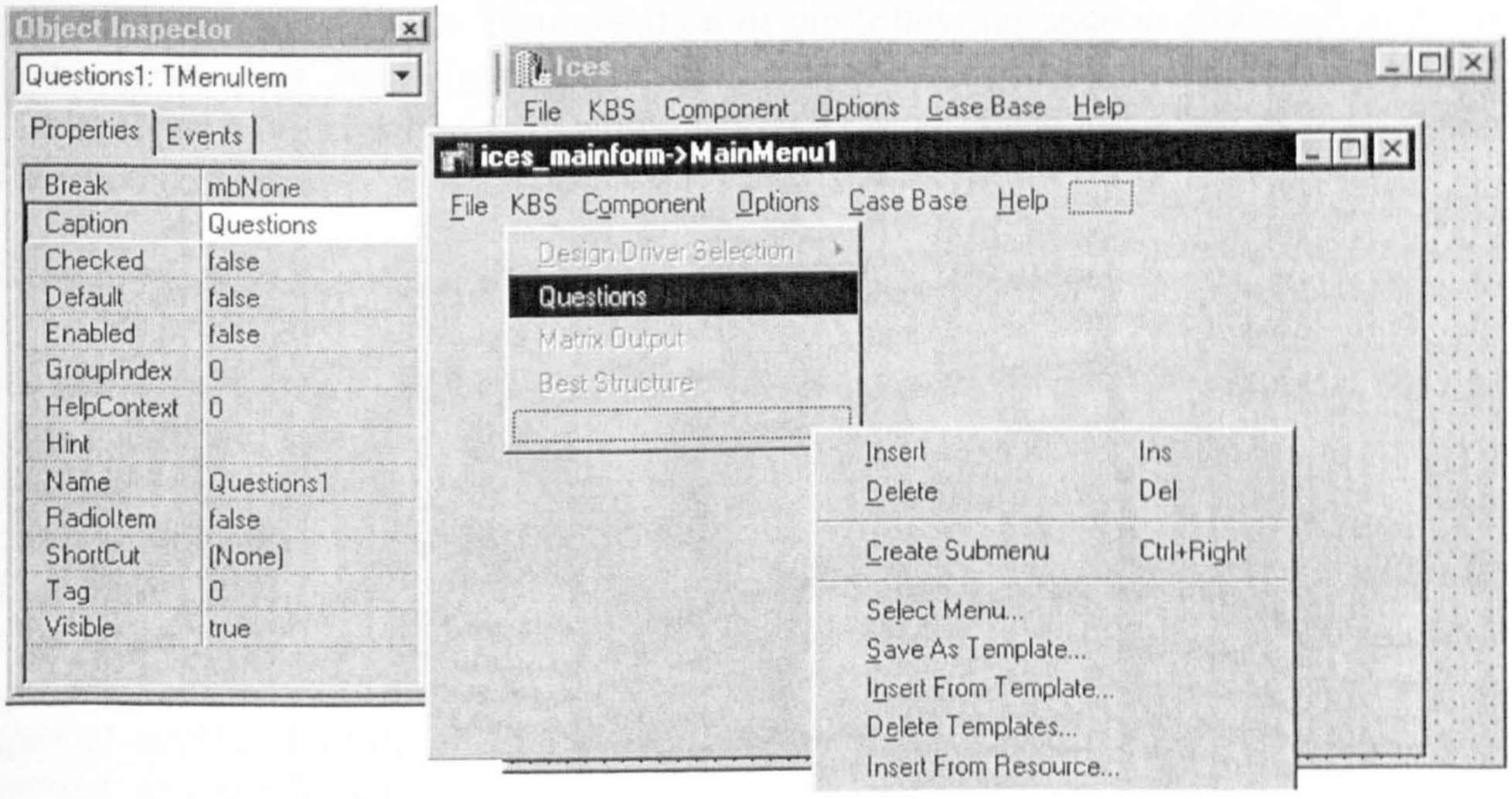



Figure C.3, Main Menu Edit Options

Firstly, select the Questions dialog box development form from the Project Manager. The Project Manager is accessed from the View main menu option in the C++ Builder development environment. The Questions dialog box development form is illustrated in Figure C.4. Space can be created in the Questions dialog box development form in which to place the text for further questions by dragging the dialog box control buttons to the bottom of the form. The developer will see that C++ Builder automatically creates scroll bars once the control buttons are dragged outside of the parameters of the existing dialog box form. Thus, increasing the amount of useable space within the dialog box. Having created sufficient space in the Questions

dialog box development form the developer should select the Label option from the C++ Builder development environment tool bar. The developer should click on the Questions dialog box development form were it is desired to place the new question text. A text component will automatically appear on the form. This component will contain default text e.g., Label13, this should be deleted by the developer and substituted with the question text. This is achieved by the developer writing the desired question in the Caption field of the Object Inspector.

In order to permit the user to give a response to the new question it is necessary add an edit box to the Questions dialog box development form i.e., to enable the entry of a character or numerical value. An Edit Box component should be selected from the development environment tool bar. Again, the developer should click on the appropriate location on the Questions dialog box development form i.e., below the existing edit boxes. The new edit box will appear. The new edit box can be sized by dragging with the cursor on its edges. The default text in the edit box is easily removed via the Object Inspector. Figure C.5 shows the Questions dialog box development form with a new question and edit box entered.

Clearly, once a new question has been entered in the Questions dialog box it is necessary to expand the existing supporting code so that the answer to the question has meaning throughout the KBS. With this in mind, the reader and/or developer is referred to section C.2.3.2. The operation of the following code is discussed in detail in this section. As such, it is proposed here only to highlight those aspects of the code that need to be expanded to accommodate additional questions being entered in the Questions dialog box.

```
void __fastcall Tquestions::okay_buttonClick(TObject *Sender)
{
int len;
float qa, qb, qc;
char buffer1[30];
char buffer2[30];
char buffer3[30];
char buffer7[30];
qa=get_number(Edit4);
qb=get_number(Edit5);
qc=get_number(Edit6);
kbs_questions.max_wall_thickness[0] = qa;
kbs_questions.question_answer5[0] = qb;
kbs_questions.question_answer6[0] = qc;
len = Edit1->GetTextBuf(buffer1,30);
len = Edit2->GetTextBuf(buffer2,30);
len = Edit3->GetTextBuf(buffer3,30);
len = Edit7->GetTextBuf(buffer7,30);

        if(buffer1[0] == 'y')
        {
        kbs_questions.homogenous_structure[0] = 1;
        }
```

```
else
{
kbs_questions.homogenous_structure[0] = 0;
}

if(buffer2[0] == 'y')
{
kbs_questions.internal_section[0] = 1;
}
else
{
kbs_questions.internal_section[0] = 0;
}

if(buffer3[0] == 'y')
{
kbs_question.load_path[0] = 1;
}
else
{
kbs_questions.load_path[0] = 0;
}

if(buffer7[0] == 'y')
{
kbs_questions.question_answer7[0] = 1;
}
else
{
kbs_questions.question_answer7[0] = 0;
}
}
```

As indicated in section C.2.3.2, the above function is invoked when the user clicks on OK button of the Questions dialog box. The purpose of this function is to assign variables to the global variable arrays residing in the *structure* QUESTION_RESPONSE which is defined in the header file *buffer.h*. The method by these variables are assigned differs according to the input required by the questions residing in the Questions dialog box. That is, input may be numerical or character.

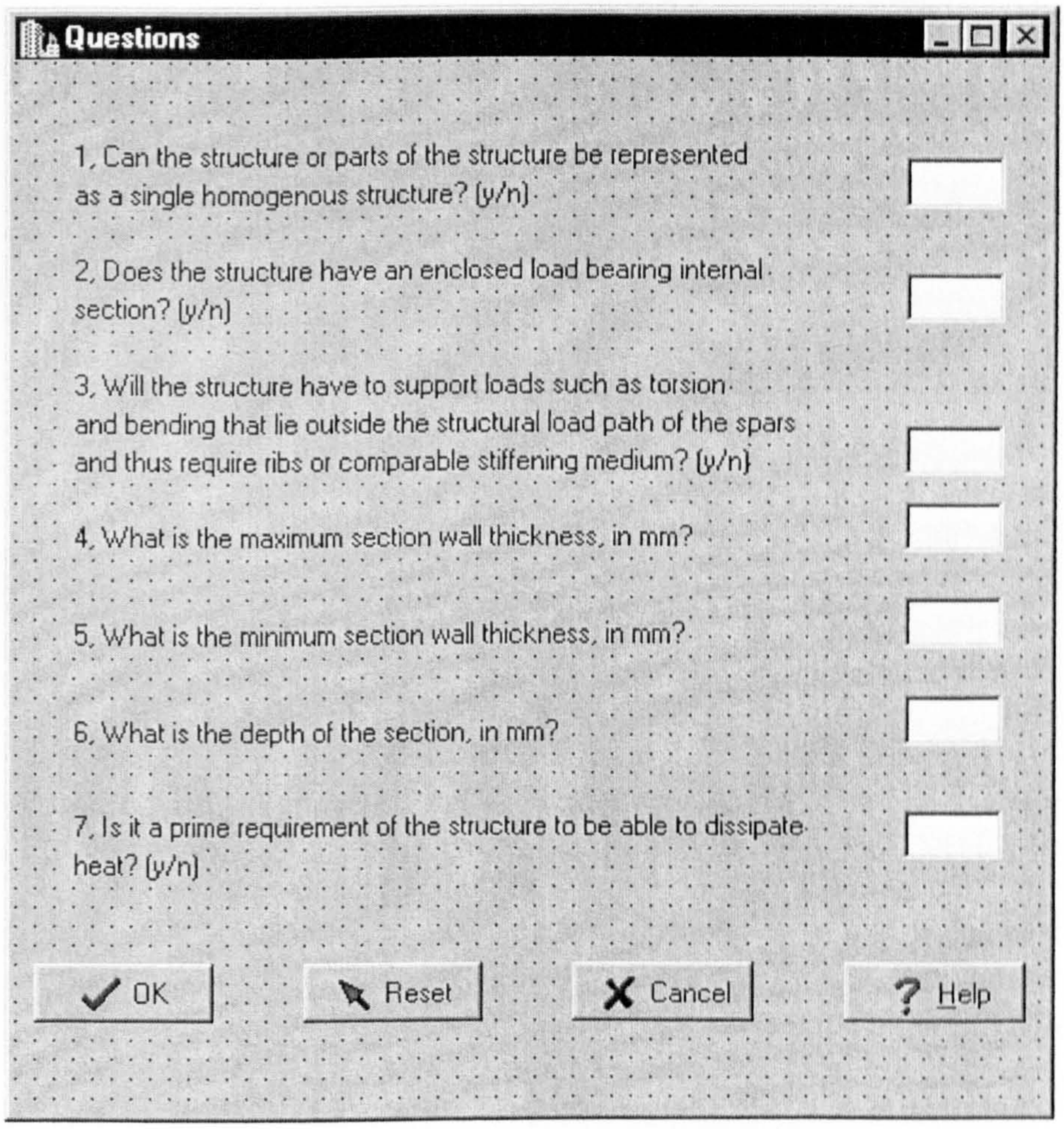



Figure C.4, Questions Dialog Box Development Form

Looking first at numerical input. For the above function to accommodate an additional numerical question it is necessary to include another float variable i.e., in addition to *qa*, *qb*, and *qc*. Further, there must be another call to the function *get_number* which takes the string in the newly created edit box to be converted into a floating point number, see section C.2.3.2 for an explanation of this function. In the context of assigning the floating point value, to the *structure* QUESTION_RESPONSE it is necessary to add an additional float array to this *structure*. Thus, enabling the variable to be assigned.

Turning attention now to character input. For the above function to accommodate character input i.e., 'y' or 'n'. it is necessary to include another character array i.e., in addition to *buffer1*, *buffer2*, *buffer3* and *buffer7*. Further, there must be another call to the function *GetTextBuf* which takes the string from the newly created edit box and assigns it to the newly created character array. It is then necessary to create additional *if* and *else* conditional statements so that the character placed in the character array can be compared to the character 'y' as in following segment of code taken from the above function.

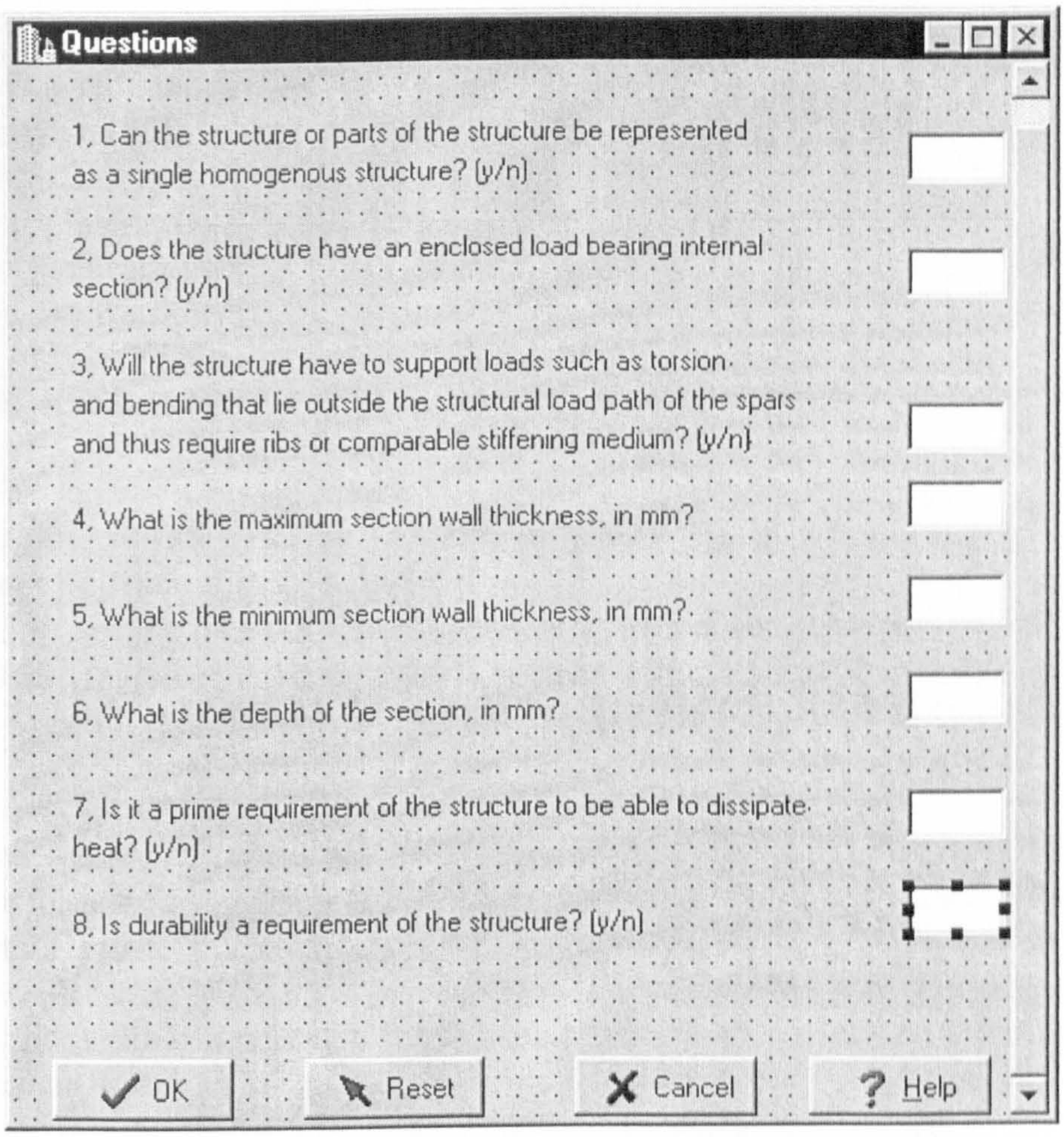



Figure C.5, The Questions Dialog Box Development Form with a New Question and Edit Box Entered

```
if(buffer1[0] == 'y')
{
kbs_questions.homogenous_structure[0] = 1;
}
else
{
kbs_questions.homogenous_structure[0] = 0;

}
```

As can be seen from this code, depending on the outcome of this comparison, 1 or 0 is assigned to the *structure* QUESTION_RESPONSE i.e., kbs_questions points to QUESTION_RESPONSE. In a similar manner to the numerical input discussed above, it is necessary to add an additional integer array to this *structure* to facilitate the entry of character input. Thus, enabling the values of 1 or 0 to be assigned. The QUESTION_RESPONSE *structure* residing within *buffer.h* in its current form within the ICES prototype is given below. Clearly, adding additional arrays to this *structure* is a simple task.

```
typedef struct
{
```

```
int homogenous_structure[50];
int internal_section[50];
int load_path[50];
float max_wall_thickness[50];
float question_answer5[50];
float question_answer6[50];
int question_answer7[50];
}QUESTION_RESPONSE;
```

### C.5.4. Expanding the Design Driver Ranking Matrix Output dialog box to support additional design drivers

As indicated in section 9.2.1.3.3 of Chapter 9 it would be desirable to be able to expand the Design Driver Ranking Matrix Output dialog box so that it may support additional design drivers. To provide this capability the procedure below should be followed.

It is first necessary to select the Design Driver Ranking Matrix Output dialog box development form from the Project Manager. The Project Manager is accessed from the View main menu option in the C++ Builder development environment. The Design Driver Ranking Matrix Output dialog box development form is illustrated in Figure C.6. Space can be created in the Design Driver Ranking Matrix Output dialog box development form in which to place further design drivers and associated edit boxes by dragging the dialog box control buttons to the bottom of the form. If the developer drags these control buttons outside the parameters of the existing dialog box form C++ Builder will automatically create scroll bars. Thus, increasing the amount of useable space within the dialog box. Having created sufficient space in the Design Driver Ranking Matrix Output dialog box development form, the developer should select the Label option from the C++ Builder development environment tool bar. The developer should click on the Design Driver Ranking Matrix Output dialog box development form where it is desired to place the new design driver text; this being most likely below the other design drivers on the form. A text component will automatically appear on the form. This component will contain default text e.g., Label27, this should be deleted by the developer and substituted with the text corresponding to the desired design driver. This is achieved by the developer writing the name of the design driver

in the Caption field of the Object Inspector.

In order to facilitate the calculation of the new design driver's importance rating it is necessary that five edit boxes be placed on the form adjacent to the new design driver's name. An Edit Box component should be selected from the development environment tool bar. The developer should click on the appropriate location on the Design Driver Ranking Matrix Output dialog box development form; this will be below one of the columns of existing dialog boxes. A new edit box will appear. This operation should be repeated four more times until there developer has created five new edit boxes. An edit box can be sized by dragging with the cursor on its edges. The default text in the edit boxes is easily removed via the Object Inspector. Figure C.7 shows the Design Driver Ranking Matrix Output dialog box development form with a new design driver entered i.e., durability, and the associated edit boxes.

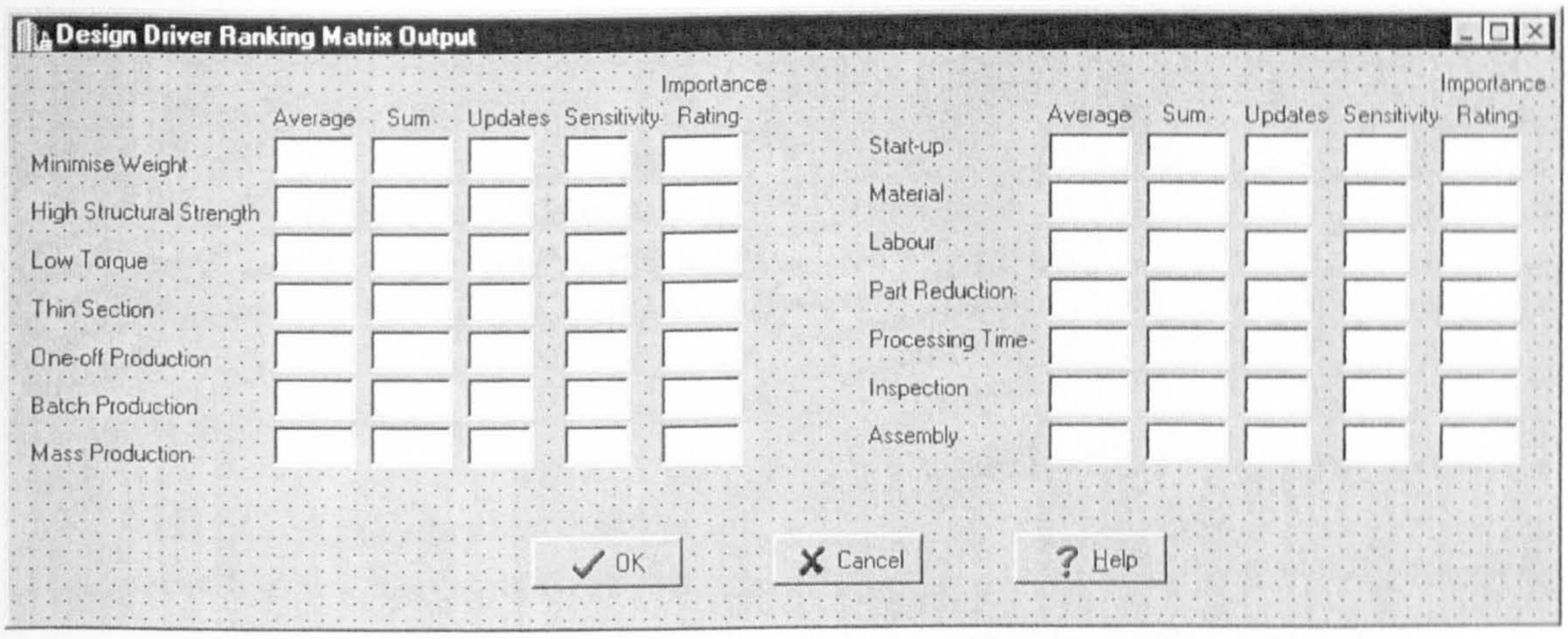



Figure C.6, The Design Driver Ranking Matrix Output Dialog Box Development Form

Clearly, once a new design driver has been placed on the Design Driver Ranking Matrix Output dialog box it is necessary to expand the existing supporting code so that the design driver's existence has meaning throughout the KBS operation. The operation of the following code is explained in detail in section C.2.3.3. As such, it is proposed here only to highlight those aspects of the code that need to be expanded to accommodate a new design driver acting on the system.

```
void __fastcall Tmatrix::FormActivate(TObject *Sender)
{
float mw_sens; //sensitivity variable
float matmwa, mwavg, mwsum, mwadd_avg_sum; //minimum weight variables
float mwsens, mwavgsum, mwupdate, mwimp_rat;

//assign minimum weight value from buffer.h to variable
matmwa=mw_info.MinWeightInformation1[0];

//assign sensitivity value from buffer.h to variable
mw_sens=sense_info.SensitivityInformation1[0];

        //if sensitivity buffer not empty continue
if(mw_sens != '\0')
{
//write contents of buffer to edit box
Edit22->Text=mw_sens;
//overwrite sensitivity buffer
sense_info.SensitivityInformation1[0] = '\0';
}

//if minimum weight buffer not empty
if(matmwa != '\0')
{
```

```
int mw_update, r, x;
r = 1;
//write contents of buffer to edit box
Edit1->Text = matmwa;

//get contents of update edit box
mw_update = get_number(Edit15);
//increment the update value by 1
x = mw_update + r;
Edit15->Text = x;

mwavg = get_number(Edit1);
mwsum = get_number(Edit8);
//calculate the new mean average value of minimum weight value scores
mwadd_avg_sum = mwavg + mwsum;
Edit8->Text = mwadd_avg_sum;

mwsens = get_number(Edit22);
mwavgsum = get_number(Edit8);
mwupdate = get_number(Edit15);

//calculate the ddrt importance rating
mwimp_rat = sqrt(((mwavgsum / mwupdate) * mwsens));
Edit29->Text = mwimp_rat;
//assign null value to buffer to ensure update value works correctly
mw_info.MinWeightInformation1[0] = '\0';
}
}
```

The above function relates to the minimum weight design driver. To enable a new design driver to operate on the system the above function should be copied. The float and integer variables should be changed to reflect the new design driver. In addition, the edit box names that correspond to the new design driver should be substituted for those that currently reflect the minimum weight design driver.

In conjunction with adapting the above function the developer will have to add an additional global variable array to the STRUCT_SEN_INFO *structure* residing in the *buffer.h* header file. This is necessary in order for the sensitivity value of the new design driver to be accessed. In a similar manner, the developer will have to introduce

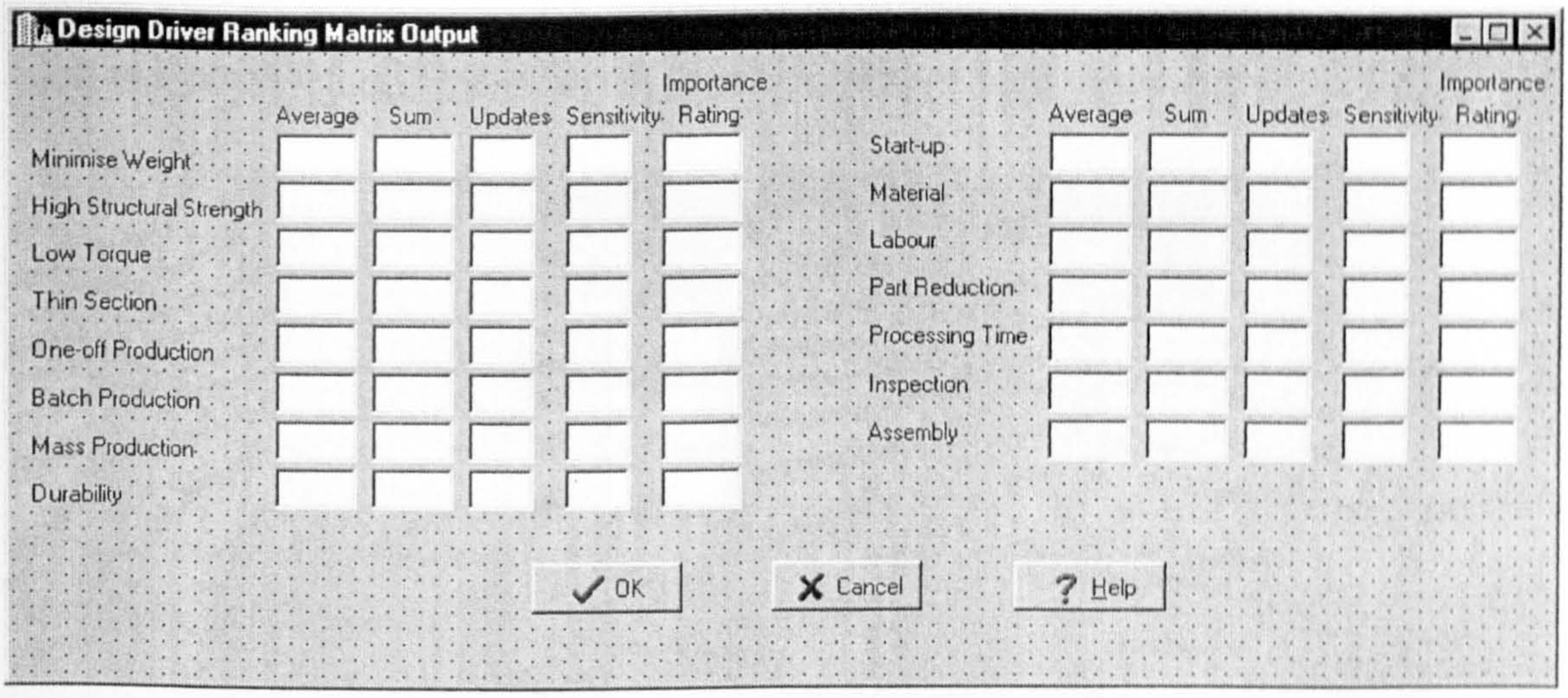



Figure C.7, The Design Driver Ranking Matrix Output Dialog Box Development Form

with a New Design Driver Entered

a new *structure* in the header file *buffer.h*. The purpose of this *structure* is to accommodate the mean average design driver impaction score for the new design driver as determined by the design driver update dialog boxes as described in section 9.2.1.4.1 of Chapter 9 and section C.2.4.1 of this appendix. If the new design driver was in fact durability it is conceivable that the new *structure* might look something similar to the following:

```
typedef struct
{
float durability_information1[50];
}STRUCT_DURA_INFO;
```

Clearly, it will be necessary to write and read the information residing in the new edit boxes, corresponding to the new design driver, to and from file. How this is achieved is explained fully in section C.2.3.3.

### C.5.5. Adding additional design driver Default Update dialog boxes to the ICES Prototype

Section 9.2.1.4.1 of Chapter 9 introduced the Default and Default Update dialog boxes. These dialog boxes show in numerical terms the impact that one design driver has on all the other design drivers operating in the system. The operation of these dialog boxes relate directly to the design driver matrix and design driver ranking technique (DDRT) as described in Chapter 7.

In terms of expanding the ICES prototype to accommodate new design drivers this appendix provides details with respect to how to make a copy of a Default Update dialog box i.e., copying it to the C++ Builder Object Repository. Thus, enabling this copy at some time in future be called up from the Object Repository, edited and subsequently used in the context of a new design driver. In addition, this discussion

provides the reader with an explanation of how to expand the existing supporting code such that new Default Update dialog box can be appended to the current ICES

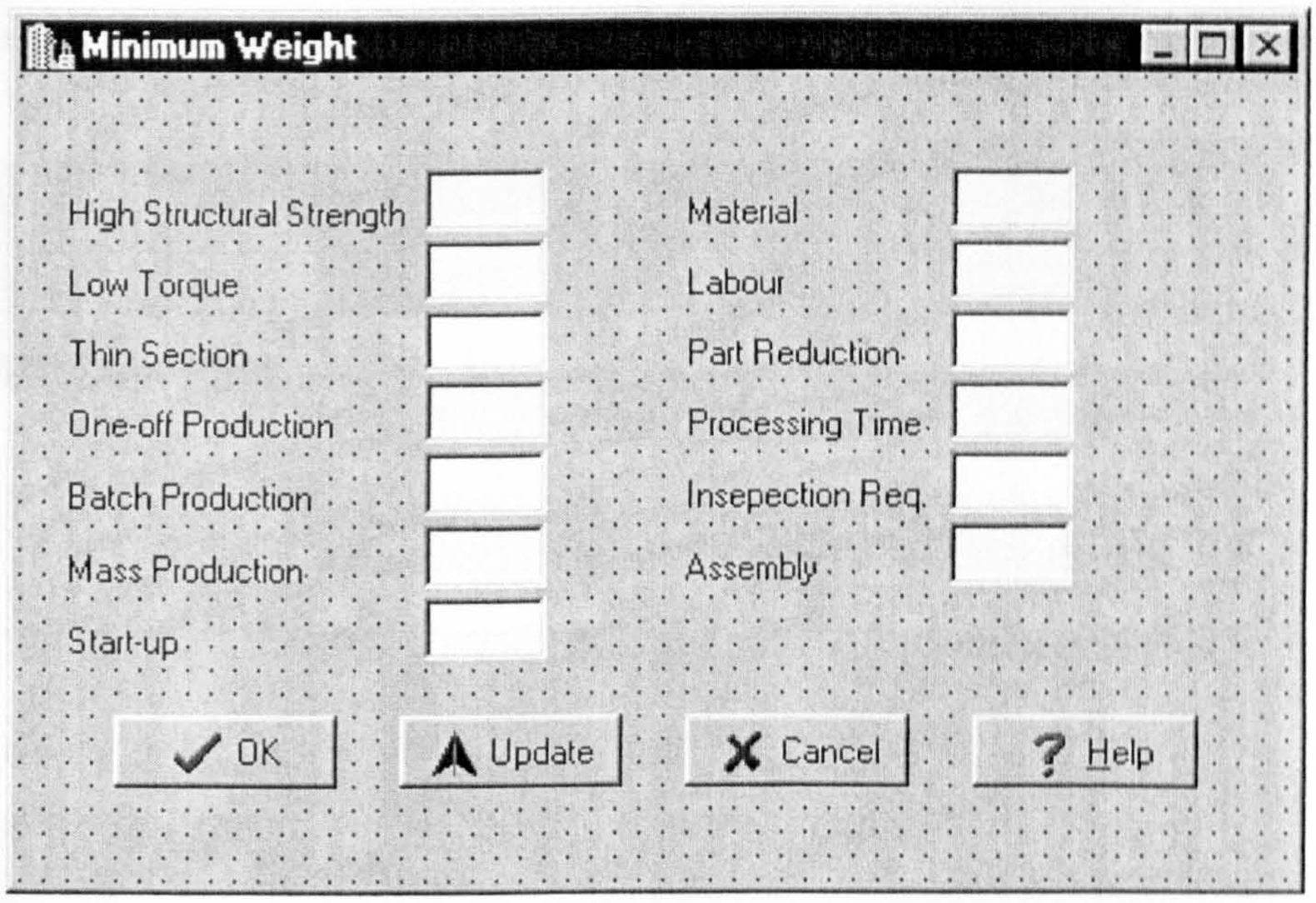



Figure C.8, The Minimum Weight Default Update Dialog Box Development Form

prototype. The Minimum Weight Default Update dialog box is the Default Update box used in this example. However, any of the design driver Default Update dialog boxes would have sufficed.

It is first necessary to select the Minimum Weight Default Update dialog box development form from the View main menu option in the C++ Builder development environment. The Minimum Weight Default Update dialog box development form is illustrated in Figure C.8. The developer should now click on the Minimum Weight Default Update dialog box development form to make it the active component in the development environment. The developer should now click on the right mouse button. This will call up a pop-up menu; from this menu the developer should select the Add To Repository menu option. The Add To Repository menu option causes the Add To Repository dialog box to appear, as illustrated in Figure C.9. It can be seen from the figure that the Minimum Weight Default Update dialog box is highlighted in the Forms list box of the Add To Repository dialog box i.e., min_weight_update. The developer should now enter the details of the Minimum Weight Default Update dialog box in the right-hand side of the Add To Repository dialog box. In the context of entering these details the Page option is the most important field in the Add To Repository dialog box. This presents a pull-down list box with a range of option i.e., Forms, Dialogs, Data Modules and Projects. These options classify the range of objects stored in the Object Repository. In this instance, the developer should select the Dialog option. Having entered all the details concerning Minimum Weight Default Update dialog box into the Add To Repository dialog box the developer should click on the OK button. The Minimum Weight Default Update dialog box has now been placed in the Object Repository.

Assuming that its now desired to create a new Default Update dialog box for a new design driver the developer can access the Object Repository and call up a copy of the Minimum Weight Default Update dialog box. This may then be edited such that it conforms to the requirements of the new design driver. To call up the Object Repository

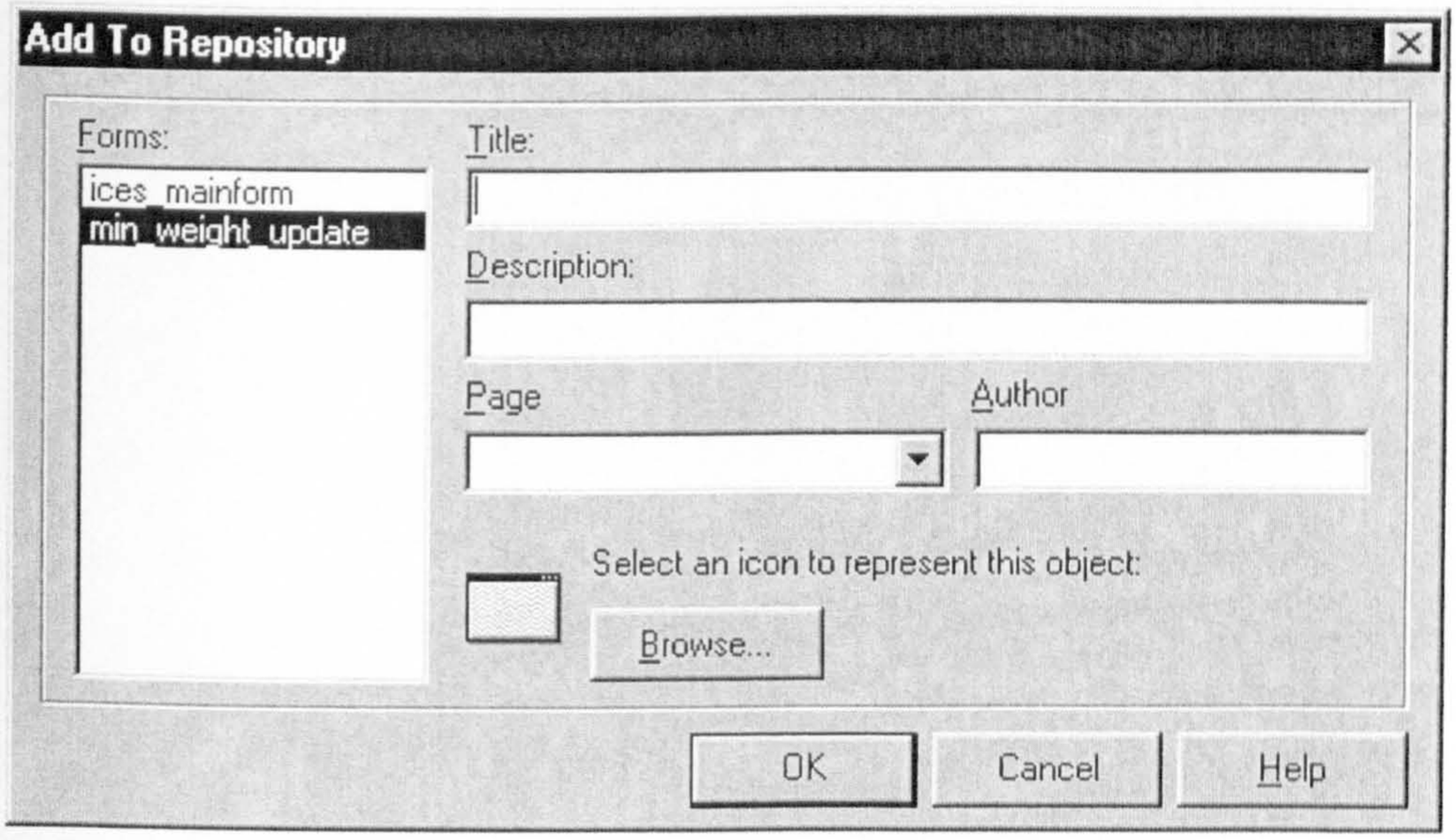



Figure C.9, The Add To Repository Dialog Box

and obtain a copy of the Minimum Weight Default Update dialog box is a simple task. The developer should select the New menu option from the File menu in the C++ Builder development environment. The selection of the New menu option causes the New Items dialog box to be displayed, see Figure C.10.


Figure C.10, The New Items Dialog Box

The developer should select the Dialogs tab at the top of this dialog box. This causes the New Item dialog box to display a series of icons which relate to all the dialog boxes stored within the Object Repository. The developer should select the minimum

weight icon with the mouse cursor and click on the OK button. This operation will cause a copy of the Minimum Weight Default Update dialog box to be presented to the user in the development environment. This copy of the Minimum Weight Default Update dialog box can now be edited in any way that the developer desires using the development tool bar and the Object Inspector. In the context of the ICES prototype this could be to facilitate the support of a new design driver. It is important to note that the copy of the Minimum Weight Default Update dialog box is a complete copy, in that, it comes with not only the visual components but also a copy of the underlying supporting code. Clearly, if the Minimum Weight Default Update dialog box is to be adapted to support a new design driver it will be necessary for this code also to be adapted also. The code below is the underlying code that supports the operation of the Minimum Weight Default Update dialog box.

```
void __fastcall Tmin_weight_update::BitBtn2Click(TObject *Sender)
{
float mwa, mwb, mwc, mwd, mwe, mwf, mwg, mwh, mwi, mwj;
float mwk, mwl, mwm, mw_total;

mwa = get_number(Edit1);
mwb = get_number(Edit2);
mwc = get_number(Edit3);
mwd = get_number(Edit4);
mwe = get_number(Edit5);
mwf = get_number(Edit6);
mwg = get_number(Edit7);
mwh = get_number(Edit8);
mwi = get_number(Edit9);
mwj = get_number(Edit10);
mwk = get_number(Edit11);
mwl = get_number(Edit12);
mwm = get_number(Edit13);

mw_total = (mwa+mwb+mwc+mwd+ mwe+mwf+mwg+mwh+mwi+mwj+
            mwk+mwl+mwm) / 13;

mw_info.MinWeightInformation[0] = mw_total;
}
```

The reader is referred to section C.2.4.1 for a detailed explanation of the operation of the above code. In the context of adapting this code for a new design driver, this may be done with relative ease. The developer must substitute the fourteen float variables in the above code for fourteen new ones which reflect the new design driver. These new variables should be substituted throughout the operation of the above function. In addition, the developer must declare a new *structure* in the header file *buffer.h*. The purpose of this structure is to accommodate the mean average design driver impaction score for the new design driver. As indicated in appendix C.5.4, if the new design driver was in fact durability it is conceivable that the new *structure* might look something similar to the following:

```
typedef struct
{
float durability_information1[50];
}STRUCT_DURA_INFO;
```

To be in a position to add a new Default Update dialog box to the ICES prototype application there is a requirement to add new menu options to the Design Driver Matrix Update menu. The adding of new menu options to the ICES prototype application is described in section C.5.2.

### C.5.6. Adding additional generic menu options and dialog boxes.

Referring the reader to section 9.2.2.2 of Chapter 9, which outlines how the meta rule base is implemented within the ICES prototype. In the context of expanding this aspect of the prototype the developer will need to be able to increase the generic menu options accessed through the Design Driver Selection menu and add the associated dialog boxes. The adding of new menu options to the ICES prototype application is described in section C.5.2. The creation of a new dialog box is now described below.

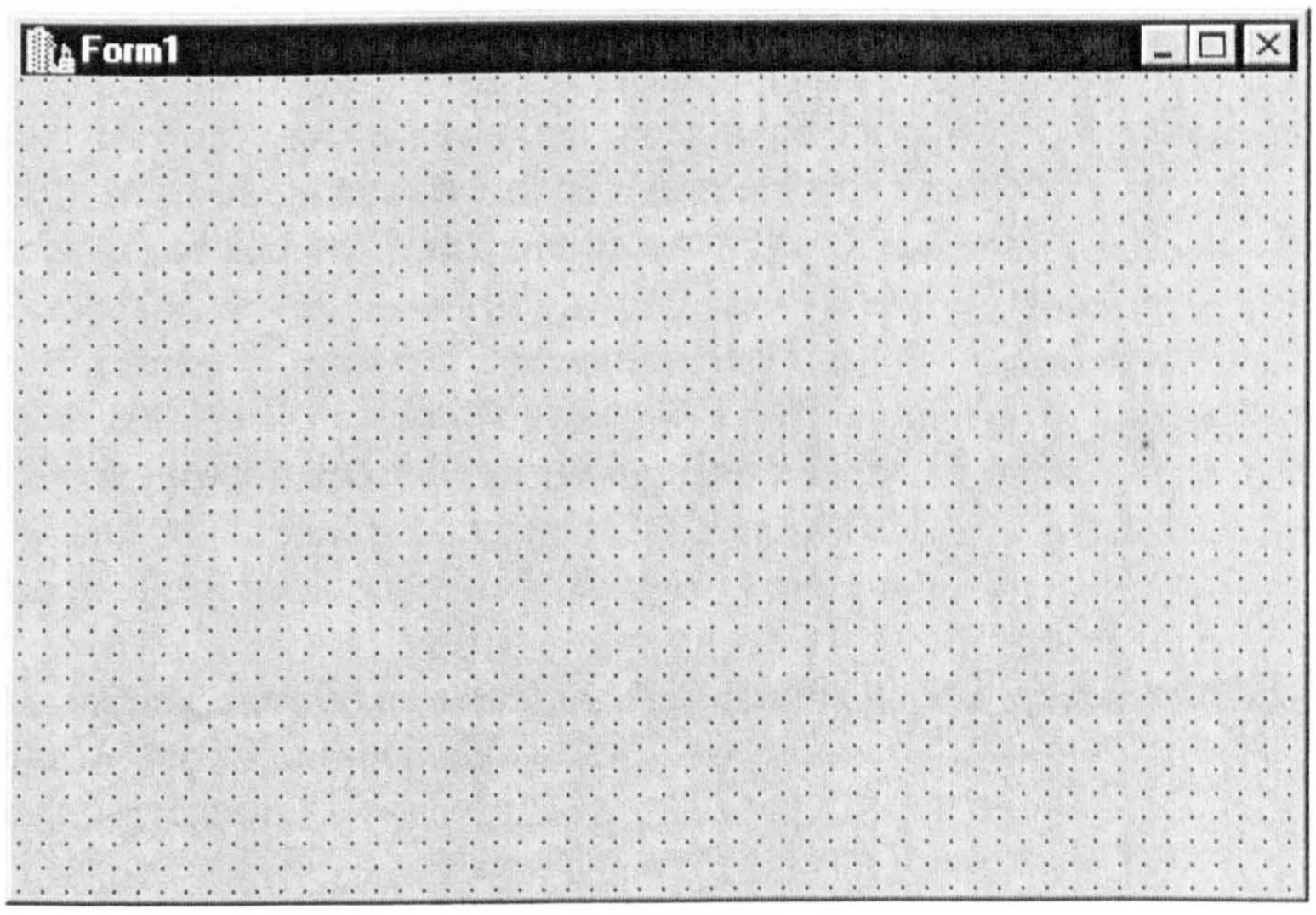



Figure C.11, A New Development Form

To create a new dialog box, the developer should select the New Form menu option or the Form icon from the File menu or the tool bar respectively in the C++ Builder development environment. On selection of either of these options a new form will automatically be placed in the C++ Builder development environment, see Figure C.11.The developer, through the Object Inspector should delete the default form caption name i.e., Form1, and insert the desired caption. This is achieved through the Caption field of the Object Inspector. The developer should then give the new form a name in the Name field of the Object Inspector. This name is the means by which the rest of the application recognises the new form.

The components necessary to convert the blank form into a recognisable dialog box should be selected from the C++ Builder development environment tool bar and placed on the form e.g. labels, push buttons, and edit boxes. Where appropriate these components may be edited using the Object Inspector e.g., removing the default text in edit boxes. It should be noted that in the context of placing components on the new dialog box, the cursor tip corresponds to the top left corner of any component to be placed. When the developer has finished building the dialog box it must be saved to file. At this juncture it will be necessary to give the new dialog box a unique file name. The new dialog box will automatically be assigned to the open project.

### C.5.7. Adding new rules to the manufacturing and structures rule bases.

An important aspect of adding any new rules to the manufacturing and/or structures rule bases is for the developer to ensure that the KBS does not become bias. That is, new rules introduced must embrace all the structural types residing in the KBS. To achieve this, the output of every new rule must be assigned to seven new individual float variables which correspond with each structural type considered by the KBS i.e., the positive correlation between the rule requirement and the design driver must be assigned to each of the new float variables corresponding to seven structural types embraced by the KBS. All new rules must be written in the function *Tb_structure* which resides in the file *best_structure.cpp* as commented in Appendix C.3.2.1.

Clearly, there are other aspects that must be consider when entering a new rule(s) into the KBS. In all likelihood, it will be necessary to further query the user. As such it will be necessary to add additional questions in the Question dialog box. This topic is covered in section C.5.3. In addition, part of the rule construction process is to determine the level of positive correlation between the rule requirement and the design driver. If there is potentially a very low positive correlation then the developer will have to construct a secondary rule to allow for this. Section C.5.8 discusses the topic of adding additional secondary rules to the KBS. To summarise, new rules entered into the KBS must possess the following components:

- A float variable must be provided for each structural type residing the ICES KBS.
- A design driver importance rating
- A correlation factor

The developer who requires a codified example of existing rules that reside in the structures and manufacturing rule bases of the KBS is referred to section C.3.2.3.

### C.5.8. Adding new secondary rules to the ICES KBS

This appendix discusses what the developer should do to add new secondary rules to the ICES KBS. If a new rule is added to the manufacturing or structures rule bases of the KBS which has a correlation between the rule requirement and the design driver which is less than 1 it will be necessary to introduce a new secondary rule as well. This secondary rule will dictate what must be done to provide a correlation between the rule requirement and the design driver which equates to 1. The KBS has fourteen dialog boxes which support the operation of secondary rules i.e., seven dialog boxes, one for each structural type in both the manufacturing and structures rule bases. Each

of these fourteen dialog boxes has a *.cpp* source file which supports the operation of the secondary rules residing in these dialog boxes. Listed below is the name of the dialog box corresponding to each structural type in each rule base and the supporting source file.

Accordion Core Structures Secondary Rules - *acc_core_struct_sr.cpp*
Accordion Core Manufacturing Secondary Rules - *acc_core_manu_sr.cpp*
Cellular Core Structures Secondary Rules - *cell_core_struct_sr.cpp*
Cellular Core Manufacturing Secondary Rules - *cell_core_manu_sr.cpp*
Spaced X-core Structures Secondary Rules - *spd_xcore_struct_sr.cpp*
Spaced X-core Manufacturing Secondary Rules - *spd_xcore_manu_sr.cpp*
Standard X-core Structures Secondary Rules - *std_xcore_struct_sr.cpp*
Standard X-core Manufacturing Secondary Rules - *std_xcore_manu_sr.cpp*
SPF Titanium Structures Secondary Rules - *spf_struct_sr.cpp*
SPF Titanium Manufacturing Secondary Rules - *spf_manu_sr.cpp*
CFC Structures Secondary Rules - *cfc_struct_sr.cpp*
CFC Manufacturing Secondary Rules - *cfc_manu_sr.cpp*
Traditional Structures Secondary Rules - *trad_struct_sr.cpp*
Traditional Manufacturing Secondary Rules - *trad_manu_sr.cpp*

The existing secondary rule dialog boxes can be expanded to support the new secondary rule text and associated check boxes in a manner similar to that described in sections C.5.3 and C.5.4. These sections discuss the adding of new questions to the Question dialog box and expanding the Design Driver Ranking Matrix Output dialog box to support additional design drivers. The only significant difference here would be that rather than selecting Edit Box components from the C++ Builder development environment tool bar, Check Box components should be selected instead.

Any new secondary rule will be a carbon copy of the principle rule placed in the manufacturing or structures rule base. The only difference being that there will now be an improved positive correlation between the rule requirement and the design driver. In addition, the output score of the secondary rule is fed to an array which is declared in the *structure* SECONDARY_RULES which resides in the header file *buffer.h*. The developer may choose to expand the *structure* SECONDARY_RULES to support the output of new secondary rules or he or she may prefer to introduce a new *structure* altogether. The developer who requires a codified example of existing secondary rules is referred to section C.3.2.7.

### C.5.9. Expansion of the ICES case base capability

This section discusses the features of the ICES case base that could be expanded in order to support additional functionality. As commented upon in Chapter 9 and elsewhere in this thesis, an aircraft foreplane was the structure chosen to drive through the ICES prototype in order to validate the methodology presented in Chapter 7. As such, a range of aircraft foreplane design cases reside in the ICES case base. However, the ICES case base is generic in nature and has the potential to support a range of aircraft structures. Examination of both the New Case and Case Query and Jury windows shows that there are no 'visual' components that are specific to the foreplane structure. Having alluded to the generic nature of the case base it is appreciated that

expansion in certain areas would enhance its functionality. As such, the remainder of this section will highlight the areas of the case base that could be expanded to support additional capability.

Clearly, it is quite conceivable that the developer may wish to introduce additional methods of querying the case base. These methods would very likely have to be operated from within a menu. As such, the developer would need to be able to further

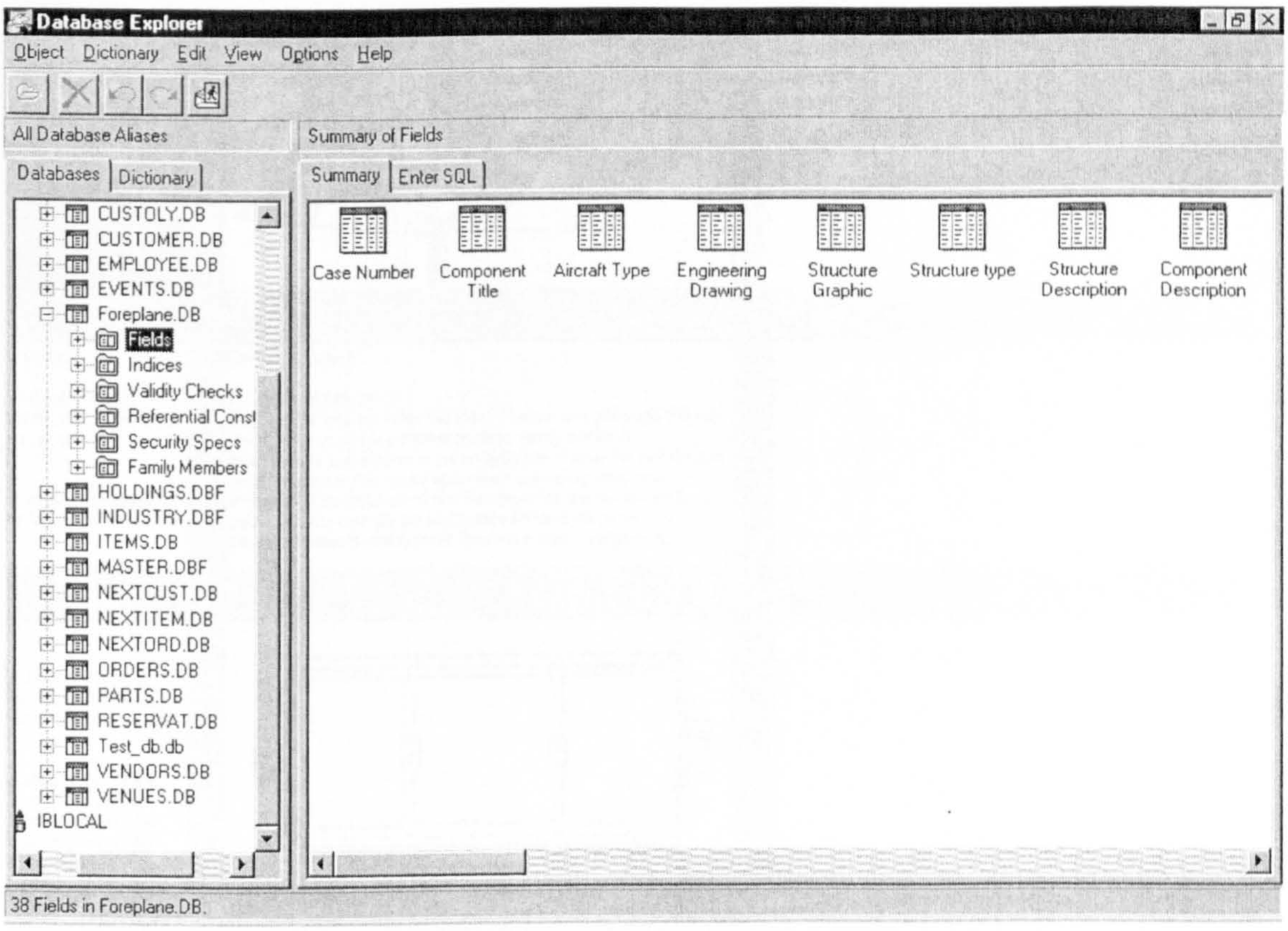



Figure C.12, The Database Explorer Dialog Box

develop the menuing capability that already exists within the ICES case base. Section C.5.2 explains how the menus operating in the case base may be edited.

At present the factual information that the case base is able to display relating to any particular design case is confined to three memo box fields and three graphic fields, plus a range of smaller edit boxes. Increasing the amount of information that the case base is able to display with respect to the design cases stored in the system is an area where the ICES case base capability could be expanded. That is, increasing the amount of information that the case base is able to display facilitates the case base being more readily able to support other aircraft structures apart from the foreplane cases presented in this study. To increase the amount of information that the case base is able to display requires that the developer increase the number fields displayed in the New Case and Case Query and Jury windows. The example now provided illustrates how it is possible to place a new field in the New Case window. The process is identical for the Case Query and Jury window.

In order to place a new field in the New Case window it is first necessary for the developer to select the Project Manager menu option from the View menu in the C++ Builder development environment. The selection of the Project Manager menu option will cause the Project Manager dialog box to be displayed. The developer should now select the New Case development form from the Project Manager dialog box. When selected, the New Case development form will be displayed in the C++ Builder development environment. With the New Case development form displayed, the developer should select the Explore menu option from the Database menu in the C++

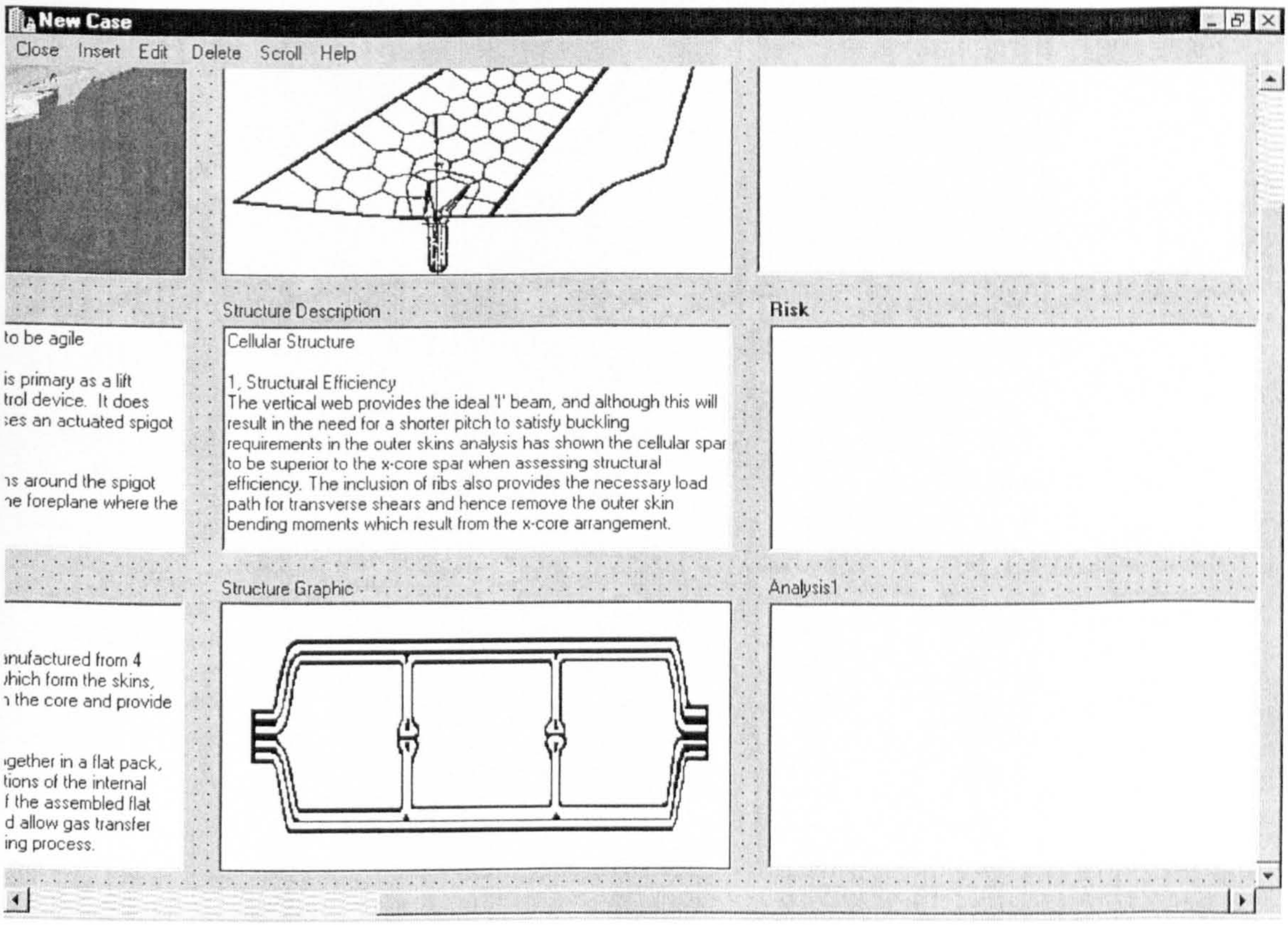



Figure C.13, The New Case Window Development Form with the New Memo Field Titled Analysis1

Builder development environment. Selection of Explore menu option will cause the Database Explorer dialog box to be displayed, see Figure C.12. From the Database Explorer dialog box the Foreplane database table and subsequently the Fields option should be selected. This will cause a tabbed window titled Summary to be displayed in the Database Explorer dialog box. In this Summary window are displayed icons representing all the fields residing in the Foreplane database table, see Figure C.12. The developer should click on the desired field icon and then click on the New Case development form where is intended to display the new field in the New Case window. The VCL component corresponding to the iconised field declared in the Summary window of the Database Explorer dialog box will be automatically be displayed in the New Case development form. This VCL component now represents a new field in the New Case window. It should be noted that when this component is placed on the New Case development form the field title will also be automatically displayed with it. This new VCL component, be it an edit box, memo box or graphic

box, can be sized as required by dragging on its edges with the cursor. As an example, Figure C.13 shows the New Case window development form with a newly created memo field with the title Analysis1. The final operation that the developer must perform is to save the changes made to the case base to file. This is achieved by the developer selecting the Save All option from the File menu in the C++ Builder development environment.

It is important to note that a new field can not be added to either the New Case or Case Query and Jury windows unless the field exists within the Foreplane database table. The Foreplane database table was created in Paradox and as such, new fields can only be added to the table via this software package. It is not proposed here to discuss how database tables are created and edited in Paradox. However, the interested reader is directed to reference 99.

In the context of adding new fields to the ICES case base capability, this has the potential to facilitate the expansion of the application of nearest neighbour matching technique. That is, it would be possible for additional design case specifications to be embraced by the technique. The code that supports the nearest neighbour matching technique as applied within the ICES case base is presented in section C.4.7 of this appendix. From the explanation provided for this code, the developer should be able to appreciate the requirements necessary for the nearest neighbour matching technique to be expanded such that it can embrace additional specifications.

# Appendix D

# Borland C++ Builder as the Development Platform for ICES

# Appendix D - Borland C++ Builder as the Development Platform for ICES

This appendix is divided into two principle sections. The first section discusses the salient features of Borland C++ Builder. The second section makes a comparison between relational and object orientated databases and their suitability with respect to being employed within the case-based reasoning (CBR) component of the second ICES prototype.

## D.1 An Overview of Borland C++ Builder

Borland C++ Builder is a RAD software product for writing C++ applications. What this means in 'layman's terms' is that C++ Builder enables the program developer to write C++ programs and also construct the accompanying graphical user interface (GUI). The GUI refers to menus, dialog boxes, main window etc. The C++ Builder package provides a drag-and-drop capability such that GUI items can be placed on base forms enabling specialised programs to be constructed. When C++ Builder is first started the user is presented with a blank Form and the integrated development environment (IDE), see Figure D.1. The C++ Builder IDE is divided into three parts. Firstly, there is the top main window. It contains the speedbar on the left and the Component Palette on the right. The speedbar provides one-click access to tasks like opening, saving and compiling projects. The Component Palette contains a wide array of components that can be dropped onto forms. Components are things like text labels, edit controls, list boxes etc. For convenience, the components are divided into groups. The second part of the C++ Builder IDE is the Form. The Form is central and immediately below the main window, see Figure D.1. The Form is where the application is built. In order to build the application components are placed on the Form. To place a component on the Form the user clicks with the cursor on the component's button in the Component Palette and then clicks again with cursor on the location on the Form where it is desired for the component to appear. Finally, below the top main window and on the left of the screen is the Object Inspector. It is through the Object Inspector that the user may modify a component's properties and events. A component's properties control how the component operates and an event is a method that is invoked in a component as a result of that component's interaction with the user.The Object Inspector usually has two tabs, one for properties and one for events. Properties events and methods are discussed in greater detail in section D.1.1.3.

### D.1.1. The Principle Features of Borland C++ Builder

Before discussing Borland C++ Builder in greater depth it is appropriate first to discuss another Borland software package, Delphi. It is from Borland's success with this package that C++ Builder was borne.[100] The legacy of Delphi residing in C++ Builder also makes a direct impact on how application software is coded in C++ Builder.

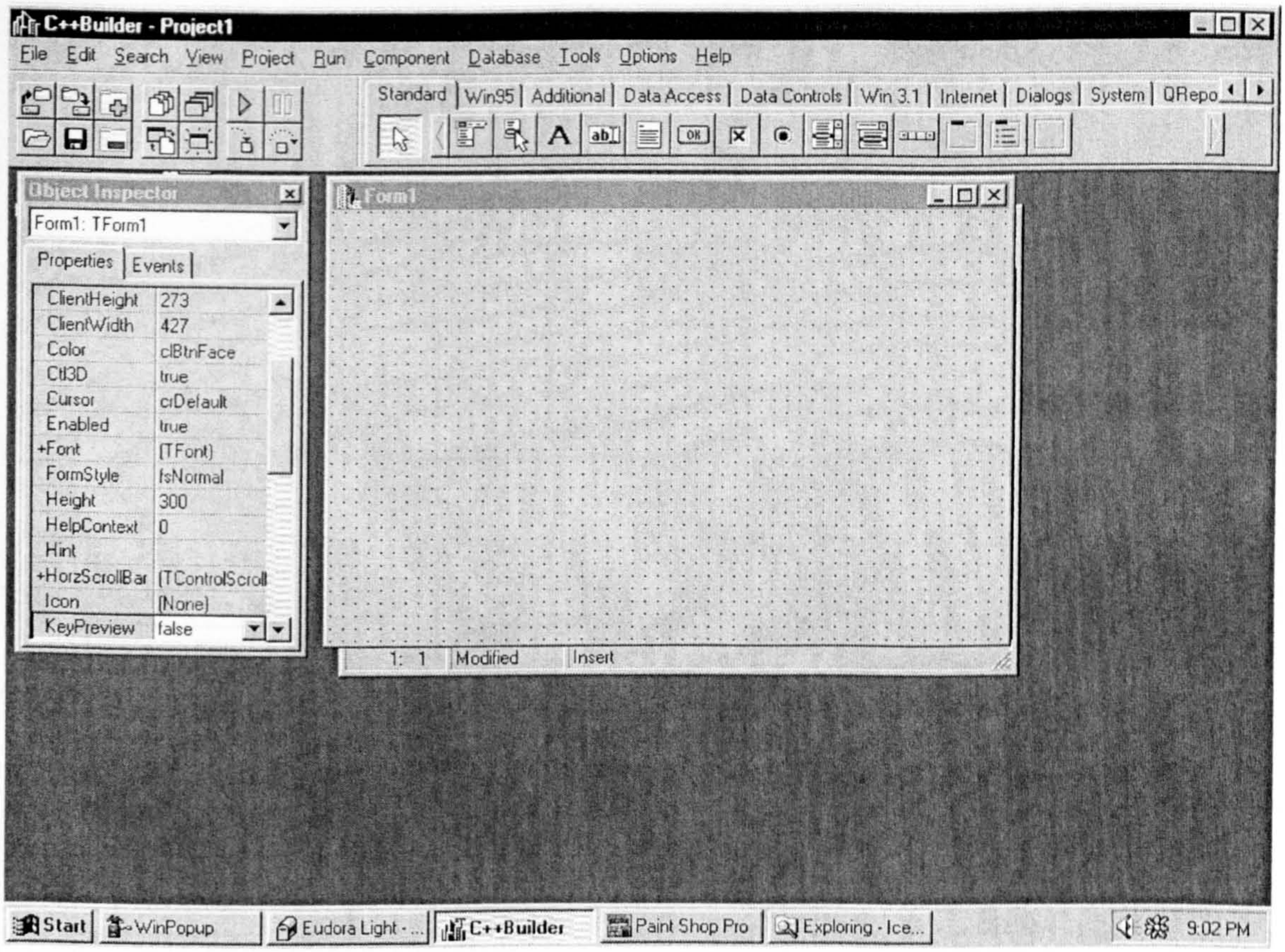



Figure D.1, C++ Builder Integrated Development Environment (IDE)

In 1995 Borland introduced a new software product called Delphi. It offered Rapid Application Development (RAD) using something called components. Components are objects that can be dropped on to a form and manipulated via properties, methods, and events. This method of producing code is know as visual programming.

The concept of form-based programming was first popularised by Microsoft's Visual Basic. Unlike Visual Basic though, Delphi used a derivative of Pascal as its programming language. This new language, called Object Pascal, introduced object oriented programming to the Pascal language. In a sense, Object Pascal is to Pascal what C++ is to C. Delphi and Object Pascal represented the marriage of object-oriented programming and form-based programming. In addition, Delphi could produce stand-alone executables i.e., real programs. Programs that did not require a run-time dynamic link library (DLL) in order to run; programs that were compiled, not interpreted; programs that ran much faster than the Visual Basic equivalent programs.

Delphi does not just rely on the use Object Pascal, it also introduced the Visual Component Library (VCL). VCL is an application framework for Windows programming in Object Pascal. This VCL which is at the core of Delphi is also at the core of C++ Builder. VCL is a library written in Object Pascal. VCL is written in Object Pascal because it was written for Delphi. Borland when developing C++ Builder decided to use the same VCL and adapt it for the C++ environment. C++ Builder has a C++ compiler and uses VCL which is an Object Pascal library. The link between VCL and C++ is seamless and enables RAD to be possible. The remainder of this section now discusses the principle features of Borland C++ Builder and highlights those

aspect which have a direct bearing on how the second ICES prototype was coded.

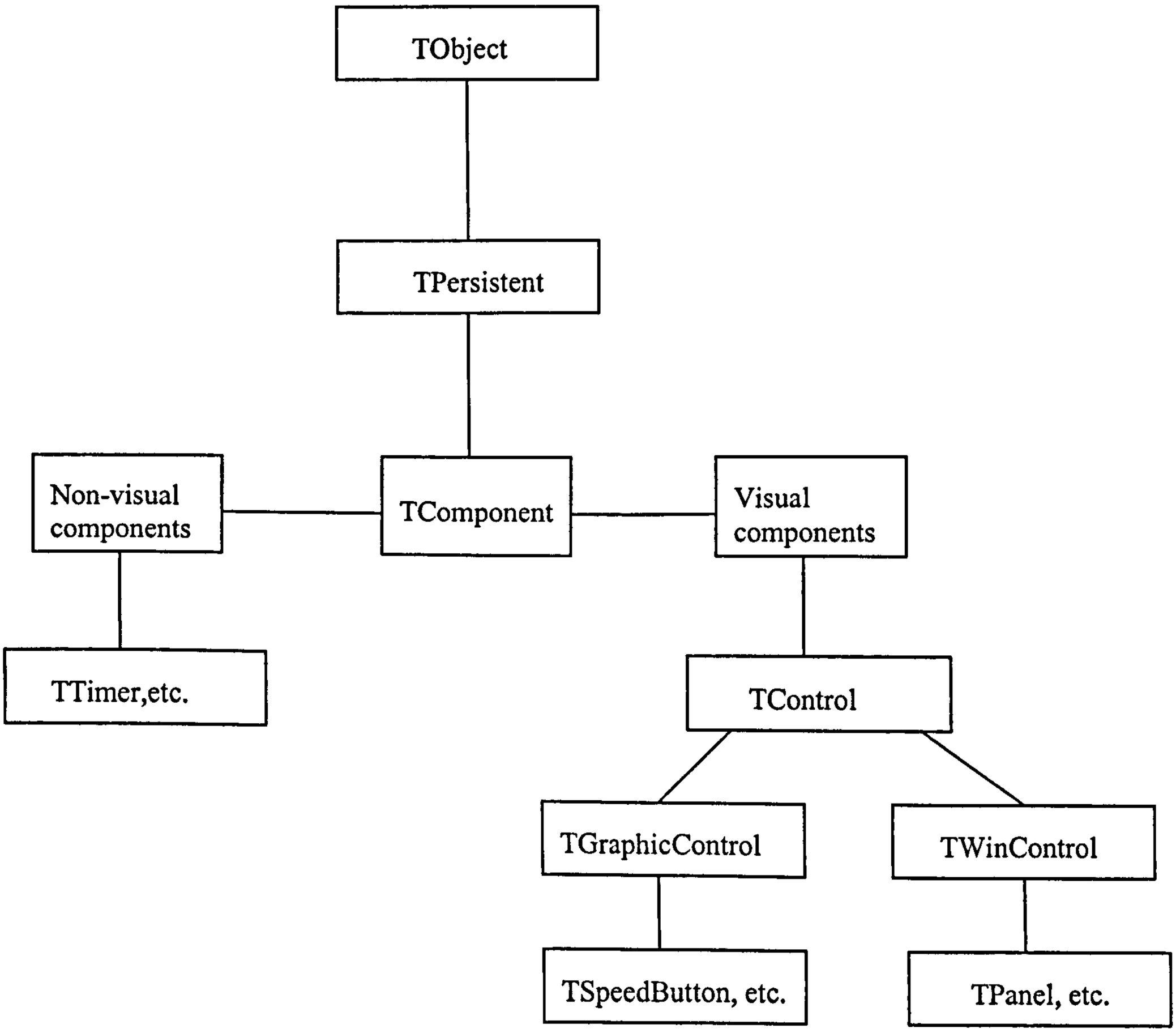



Figure D.2, Main VCL Base Classes and Derived Classes

As indicated Borland C++ Builder uses VCL. VCL is a framework which makes programming easier by encapsulating difficult Windows programming tasks into classes. These classes enable Windows programming to be approached in a rational manner. VCL provides the user with higher level programming objects that can more easily be incorporated into applications. The VCL framework makes maximum use of inheritance. The majority of the VCL framework is made up of classes that represent components. Those VCL classes which are not related to component classes perform 'house-keeping' duties i.e., act as assistant classes or provide some form of utility service. The VCL class hierarchy dealing with components is reasonably complex. Figure D.2 shows some of the main VCL base classes and some of the classes that are derived from them. The TObject is at the root of all component classes in VCL. Below TObject is the TPersistent class. This class deals with a component's ability to save itself to files and to memory. The TComponent class serves as a more direct base class for components. This class provides all the functionality that a basic component requires. Non-visual components are derived from the TComponent class. Visual components are derived from the TControl class; the TControl class provides the

additional functionality that visual components require. The individual components are then derived from either the TGraphicControl or TWin Control classes.

#### D.1.1.1. Form and Application Classes

Form and application classes are derived from TComponent and are components themselves. It is appropriate to discuss them separately in order to distinguish them from controls that are placed on a form.

##### D.1.1.1.1. TApplication

The TApplication class encapsulates the basic operations of a Windows program. The TApplication class takes care of things like managing the application's icon, providing context-sensitive help, and doing basic message handling. Every C++ Builder application has a pointer to the TApplication object called Application. The TApplication class is used primarily to execute message boxes, manage context-sensitive help, and set text for buttons and status bars.

##### D.1.1.1.2 TForm

The TForm class encapsulates forms in VCL. Forms are used for main windows, dialog boxes, secondary windows, and any other window type that it is possible to visualise. TForm is the 'work horse' class in VCL. It is in application development terms, the base class on which everything else sits. The TForm class has nearly 60 properties, 45 methods, and about 20 events which it can respond to. Properties, methods and events are discussed in section D.1.1.3.

### D.1.1.2. Component Classes

VCL component classes encompass a wide range of different classes. These classes are readily categorised. The class categories which have been utilised during the development of the ICES prototype are as follows:

- **Standard Component Classes** - The standard components are those components that encapsulate the most common Windows controls. The standard component classes include TButton, TEdit, TListbox, TMemo, TMainMenu, TCheckBox, TRadioButton, TRadioGroup and TPanel. As an example of these Windows control components consider the TMainMenu component. At design time, double clicking on the MainMenu component's icon brings up the Menu Editor. This tool enables the user to generate the desired main menu layout. The TMainMenu has properties that control whether a menu item is greyed out or not. Each menu item generated has a single menu event, OnClick, which enables the user to attach a function to the menu item selected.
- **Common Dialog Classes** - Windows has common dialog boxes for things like opening files, saving files, choosing fonts, and choosing colours. VCL encapsulates these common dialog boxes in classes representing each type. The classes are TOpenDialog, TSaveDialog, TFontDialog, TColorDialog, TPrintDialog, and TPrinterSetupDialog. VCL also adds the TFind Dialog and TReplaceDialog classes to this group of components. This group of components are non-visual in that they do not have a design-time interface that is visible to the user. However, the dialog boxes are visible when called at run-time.

- **Database Component Classes** - VCL has a large number of database components, which include both visual and non-visual components. The non-visual database components include TDataSource, TDatabase, TTable, and TQuery. These classes encapsulate the behind-the-scenes database operation and provide the software links between the user's application code and a proprority database. The visual database component classes are the part of the VCL database operations that users can see and interact with. For instance, a TDBGrid component is used to provide users with access to a database table that might be represented as a TTable component. In this manner, the TDBGrid acts as the interface between the user and TTable. The TDBNavigator component provides buttons that permit the user to move through a database table. This class includes buttons for next record, previous record, first record, last record, cancel edit, accept edit, and undo edit. Other visual database component classes link standard Windows controls to database fields. These classes include TDBText, TDBEdit, TDBListBox,and TDBImage among others.

It is important to note that VCL provides many more component class categories than discussed above. In the context of this study it is inappropriate to introduce them all here. However, the reader who is interested in the remaining class categories is referred to the Borland C++ Builder user manuals.[101]

### D.1.1.3. Properties, Methods, and Events

Properties, methods and events make up the public interface components in VCL i.e., the part of a component that the user can see and interact with. As indicated above properties are elements of a component that control how the component operates. Many components have common properties. All visual components have, for example, a Top and a Left property. These two properties control where the component will be positioned on a form. All components have an Owner property, which VCL uses to keep track of the child components a particular parent form or component owns.

A component's properties are displayed to the user in the Object Inspector. Figure D.3 shows the component properties of a standard button component. The component's properties are arranged in alphabetical order. If more properties exist than can be displayed at one time, the Object Inspector will have a scrollbar so that the user can access the additional properties. It should be noted that properties are more than simple data members of a class. Each property has an underlying data member associated with it, but the property itself is not a class data member. Changing a property often leads to code executed behind the scenes. It is possible to change properties in two ways. Firstly, properties can be changed at design time, when the form is being designed, and secondly, properties may be changed at run-time when the program is running through the written code.

Methods in VCL components are functions that can be called to make the component perform certain actions. For example, all visual components have a method called Show(), which displays the component, and a method called Hide(), which hides the component. Methods in VCL can be declared as *public*, *protected*, or *private* just as functions in C++ can be public, protected or private. These keywords mean the same in Object Pascal classes as they do in C++ classes. Public methods can be accessed by users of the component; protected methods cannot be accessed by users of the component, but can be accessed by classes (components) derived from a

component; private methods can be accessed only within a class itself. Again similar to C++ functions, some methods take parameters and return values, and some do not.

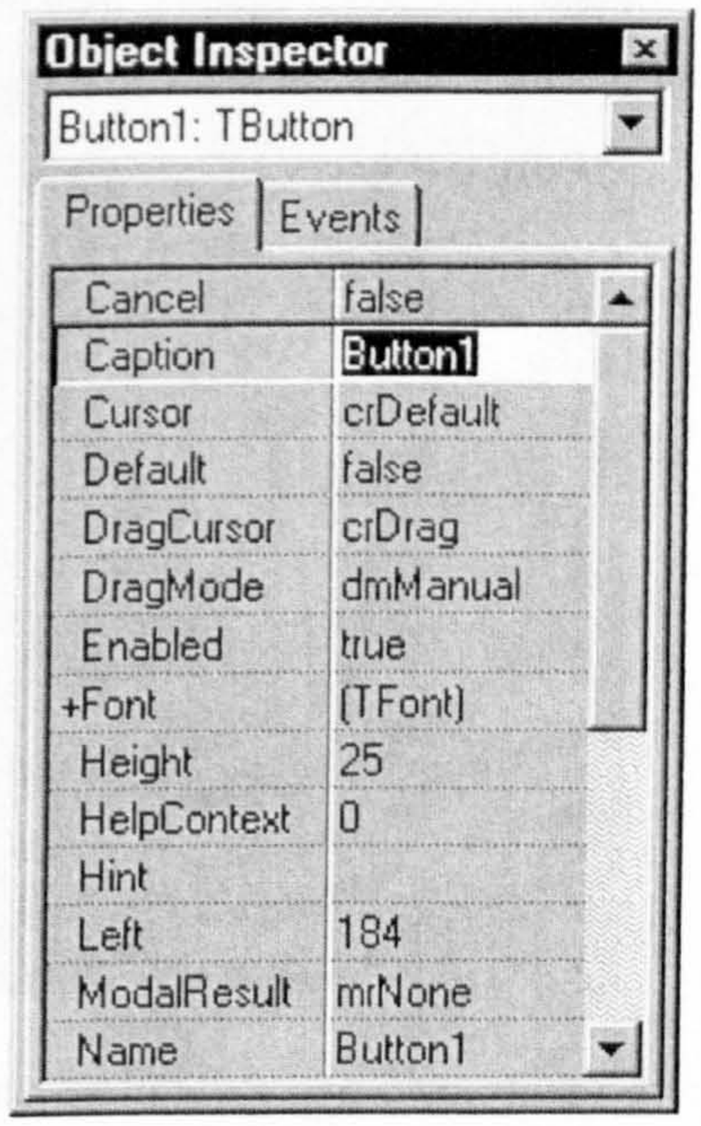



Figure D.3, The Object Inspector Showing Component Properties

The Windows programming environment is significantly different from 'MS-DOS type' programming environment. Windows is an *event-driven* environment. A Windows program (such as one developed with C++ Builder) continually polls Windows for events. Events in Windows include a menu being activated, a button being clicked, a window being moved, a window needing repainting, a window being activated etc. Windows notifies a program of an event by sending a Windows 'message'.

In VCL, an event is anything that occurs in a component that the user might need to know about. Each component is designed to respond to certain events. These events are usually Windows events but components are capable of responding to certain non-Windows events in specific circumstances. For example, a button component is designed to respond to a mouse click which is a Windows event but a database component might respond to non-Windows events such as the user reaching the end of the database record table.

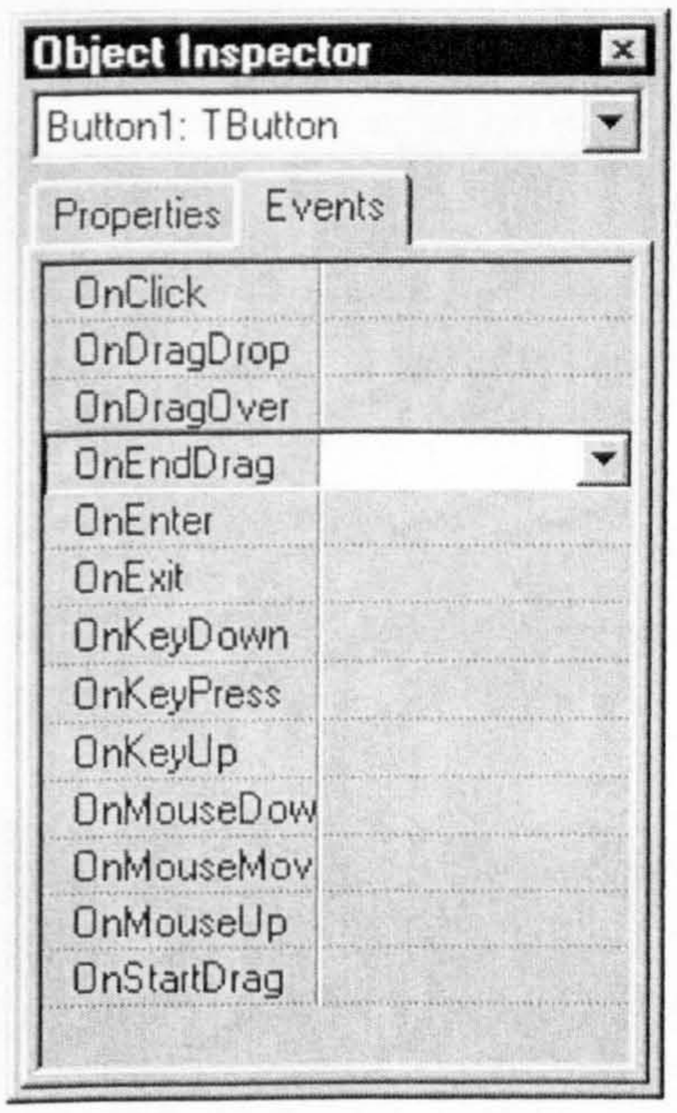



Figure D.4, The Object Inspector Showing Component Events

When a program responds to a component's event it is said to *handle* the event. Events are handled in a program through functions called *event handlers*. As part of C++ Builder's RAD environment VCL makes the handling of events very simple. The events that a component has been designed to handle are listed under the Events tab in the Object Inspector window, see Figure D.4 above. An event name is descriptive of the event to which it responds e.g., the event to handle a mouse click is called OnClick. To initiate an event and thereby generate the associated event handler the user double clicks on the event which he or she wants to initiate. Having double clicked on the desired event the associated event handler function is automatically generated in the C++ Builder IDE, see Figure D.5.

Once the event handler has been created, the user can now place his or her code in the function depending on what operation is required to take place when the event is called e.g., perform some calculation or possibly cause a dialog box to be displayed if certain prior conditions are fulfilled. Clearly, by removing the requirement for the user to physically code the more mundane function calls and associated functions considerable programming time and effort is saved. It should be emphasised however, that the user will still have to generate functions manually from time-to-time where certain operations can not be handled by VCL event handlers.

### D.1.1.4. The Limitations of Borland C++ Builder in the Context of Developing the ICES Prototype

During the coding of the ICES prototype the C++ Builder architecture imposed only one significant limitation. This being that VCL, which is Object Pascal, does not support multiple inheritance like C++. What this meant in terms of coding the second ICES prototype was that it was not possible to create a new component derived from two existing components. In addition, it was not possible for a component to support

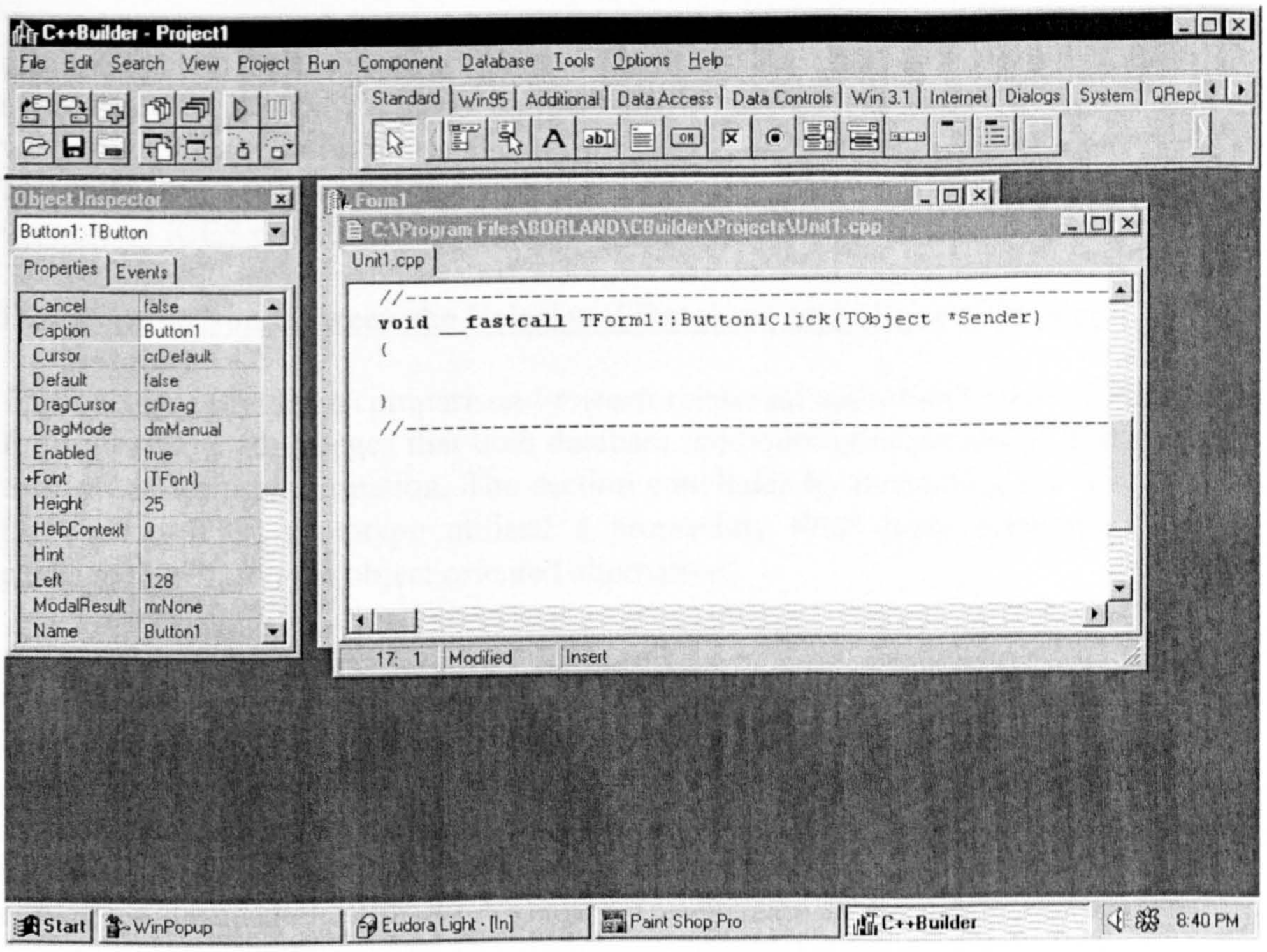



Figure D.5, Event Handler Function Automatically Generated in C++ Builder IDE

inheritance from a user generated C++ class as this would constitute multiple inheritance. The inability to derive new components from existing component classes was not a problem as C++ Builder provides such a wide range of on-board components that it is difficult to envisage a situation where it would desirable to generate a completely new component. However, the inability of a component to support inheritance from a user generated C++ class while not an insurmountable problem did severely limit the degree to which inheritance could be used throughout the coding of the prototype. Rather than pass data from one class to all those component classes that had a vested interest in the data it was necessary to make significant use of globally accessible structures in which variable arrays were nested. Component classes were made capable of reading and writing variables in and out of these variable arrays. As data could not be passed directly into a component class (through normal inheritance procedures) it was necessary to provide the component class in question with access to the relevant global structure. When the component class 'sees' that a particular variable is present in a global variable array, this acts as a 'flag' permitting the initiation of a particular action by that particular component. This could be the reading of the global variable itself so that it can subsequently be used within the component or possibly it may initiate some other action such as the operation of another dialog box, i.e., another component. The use of globally accessible structures is illustrated in Chapter 9 where the implementation of ICES in C++ Builder is discussed in detail. It is important to stress that at no time during the programming of the ICES prototype did the limitations of C++ Builder restrict the Author's capability to transfer the methodology as outlined in Chapter 7 into code.

The above discussion has provided a brief overview of the salient features of C++ Builder. Clearly, there are many more features of C++ Builder than have been

discussed here. In the context of this thesis, it was not considered necessary or appropriate to explain the software package in any greater depth. For readers who wish to gain more information about C++ Builder they should refer to reference 102. The following section now discusses the database options available when developing the second ICES prototype.

## D.2. Comparison between the Relational Database and Object Oriented Database

This section provides a comparison between relational and object oriented databases. It discusses the advantages that both database methodologies present to the user with respect to storing information. The section concludes by presenting the reasons why the second ICES prototype utilised a proprietary third party relational database package rather than the object oriented alternative.

### D.2.1. What is an Object Oriented Database?

Object oriented databases are significantly different from relational databases. The development of object oriented databases was a natural progression from work performed on object oriented languages. Just as object oriented languages attempt to more accurately model the real world represented by a program, object oriented databases try to do the same. For some applications, using an object oriented database can offer significant benefits for development effort and system performance.

#### D.2.1.1. Ability to Store Objects

The most obvious difference between an object oriented database and a relational database is the object oriented database's ability to store objects. Objects are particular instances or occurrences of a class. When using an object oriented database, you can directly store an object without first converting the object's member variables into columns in a relational table. The object oriented database approach provides a more accurate model of the real world. By directly storing objects in the database, you avoid the artificial concept of mapping class member variables to database columns.

#### D.2.1.2. Inheritance

Another distinguishing trait of an object oriented database is support for inheritance. Inheritance is the ability to define a new class by basing it on an existing class and extending the original class with additional functionality. If classes are stored in an object oriented database it is important for the database to support inheritance so that when the program tells the database to store a new object, all the attributes of that object are stored, even those which are inherited from another object. Without direct support for inheritance a program using an object oriented database would need to tell the database to store all the object's attributes right up to the top of the inheritance hierarchy.

Just like early versions of C++ many object oriented databases support only single inheritance. Multiple inheritance is a relatively recent addition to the C++ language, and it adds definite complications to object oriented databases. For example, if a subclass inherits an attribute or function with the same name from more than one superclass, this must be resolved at runtime. Because of this type of complication, many object oriented database vendors have not yet tackled the problem of multiple inheritance. Related to an object oriented database's support for inheritance is its

support for polymorphism. Polymorphism refers to the ability of an object to be more of one type. Polymorphism is discussed in sections D.2.2.1 and D.2.3.1.

### D.2.2. How Does an Object Oriented Database Work?

Before looking at how an object oriented database stores data, it is first appropriate to quickly review how a relational database stores data. As shown in Figure D.6, relational databases store data in tables. A table can be joined to another table through the use of primary key/foreign relationship. Figure D.6 shows a one-to-many relationship since each department can have more than one employee. In this example, the Design Department has two employees (Joe Smith and Mary Jones), and the Structures Department has only one (Tim Johnson).

If the reader were to move this data from the relational database of Figure D.6 into an object database, one way of doing so would be to define three classes: Department, Programmer, and Optimisation. Figure D.7 shows the same sample data, but stored in an object oriented database this time. The first line of each block in Figure D.7 indicates the type of object. The items below the first line are the names and values of each attribute of the object. For example, it can be seen that there is a Programmer whose name is Joe Smith and whose language is C++.

| Employee Table | | | |
|---|---|---|---|
| Employee ID | Name | Title | Department Name |
| 1234 | Joe Smith | Programmer | Design |
| 1235 | Mary Jones | Programmer | Design |
| 1236 | Tim Johnson | Optimisation | Structures |

| Department Table | |
|---|---|
| Name | Code |
| Design | Des100 |
| Structures | Str100 |

Figure D.6, Relational Database Table Showing a One to Many Relationship

One immediately apparent difference from the relational view of this data is that an attribute is allowed to take on more than one value. In this example, the Members attribute for the Design Department contains both Joe Smith and Mary Jones. In a relational database this type of one-to-many relationship would have been modelled as two tables sharing a primary key/foreign key dependency.

One problem with the object oriented database shown in Figure D.7 is that it seems to store data redundancy. The name of each employee is stored in the Programmer or Optimisation object and also in the Department object the employee belongs to. This is where object identifiers (OID) come into play. Figure D.8 is fairly simple, however, values should not be used to identify objects contained by an object e.g., the way Members are contained in Department in this illustration. This is because an object database does not know which attribute values, if any, uniquely identify an object. As such, an object database will assign an OID to each object and use this value to uniquely identify the object. This can be seen in Figure D.8

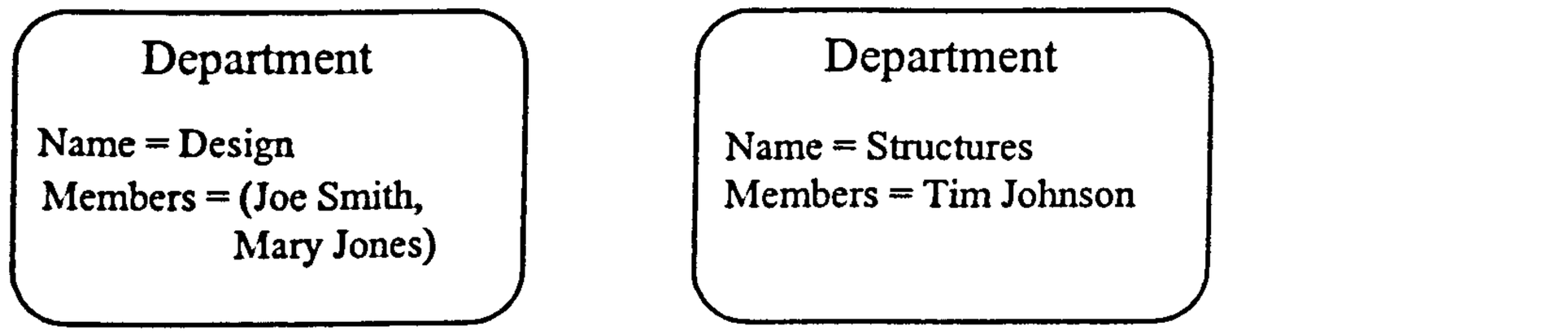



Figure D.7, The Object Oriented Database

Figure D.8 shows that the Department object, known to the database as OID 100, contains Members whose OIDs are 200 and 201. By examining Figure D.8, you can determine that an OID of 200 refers to Joe Smith and an OID of 201 refers to Mary Jones. The OIDs assigned to each object and shown in Figure D.8 follow an easily perceived pattern: OIDs assigned to Department objects start at 100 and are incremented by one; OIDs assigned to Programmer objects start at 200 and are presumably incremented by one also. When using a real object database, OIDs will typically seem like meaningless random numbers and do not enable inferences to be easily made.

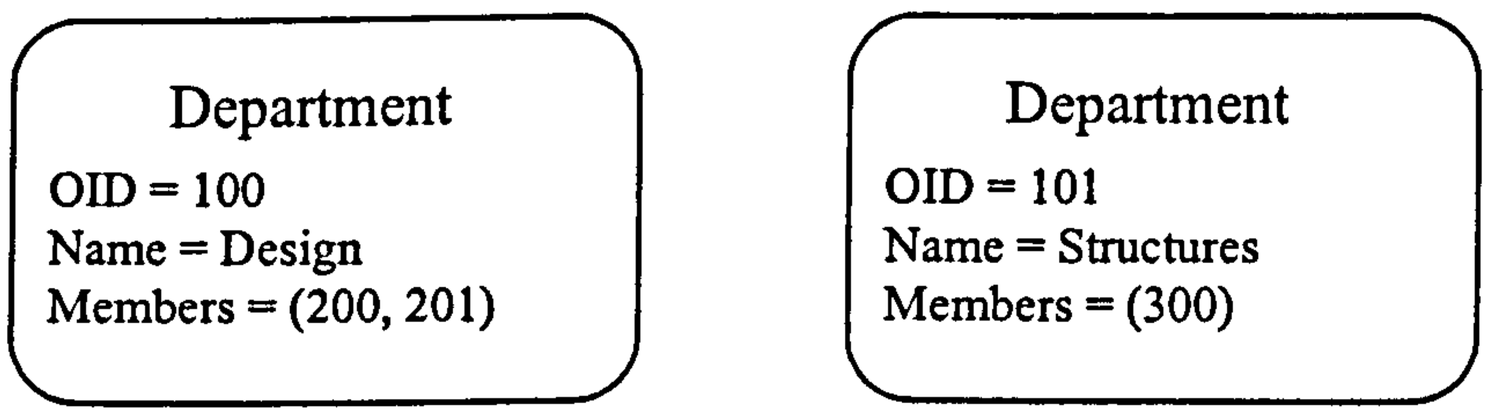



Figure D.8, The Object Oriented Database Supporting Object Identifiers

### D.2.2.1. Aggregate Objects

When working with a relational database, you are restricted to working with whatever data types are supported by the database. Typically, this is limited to types that represent simple values, such as integers, character arrays, floating-point numbers, currency, dates, and times. Not only does an object oriented database allow you to store objects composed of these simple types, it also allows you to store objects that are aggregations of other objects. For example, you could design a database that included classes for Project, Programmer, and Department. Among others, a Department has as its attributes a Project and a set of Programmers, as follows:

```
class Department
{
        protected:
                Project project;
                Programming_group programmers;
                . . . .
};
```

When a Department object is stored in the database, all its components are stored in the database as well; even those attributes that are objects themselves. This is an important difference between object oriented and relational databases. It is also responsible for some of the power inherent in an object data model. Conceptually, it is not difficult to store a Project attribute as part of a Department object in a database. However, how does an object database store a pointer to a Project? Consider the following alternative definition of a Department.

```
class Department
{
        protected:
                Project *project;
                Programming_group programmers;
                . . . .
};
```

The distinction could be desirable in a database where a single Project is the responsibility of many Departments. One possible way to store the Project pointer is to store in the database the memory address at which the project is located. Unfortunately, this would mean that when the Department object is read from the database, the project value would need to be stored again at the same memory address. Because it is very unlikely that the memory address will be available, this solution will not work. However, the solution that does work comes in the form of the object identifier assigned to each object in the database. If the object identifier of the pointed-to object (a Project) is stored as the project pointer, then when a Department object is read from the database, the referenced Project can also be read and stored randomly in memory and the address stored correctly in the project pointer.

### D.2.3. Why Use an Object Oriented Database?

With so many good relational databases in existence, and so much investment in terms of time, money and years of experience behind them, why should one even consider using an object oriented database? The following sub-sections will cover some of the advantages to using an object oriented database.

### D.2.3.1. Support for Non-Native Types

The most obvious advantage to using an object oriented database is the reason that most object oriented databases were created: the ability to store user-defined types, or classes, and easily manipulate them. If you are using an object oriented language such as C++ and also using a relational database, there are inherent complexities involved in storing an object in a two-dimensional relational database.

An object oriented database allows you to store and directly manipulate your own data types. Relational databases typically support column types, such as integer, character array, float, currency, and maybe BLOBs. With an object oriented database, you can use a database that knows about all of these, plus Invoices, Person, Manager, Employee, Video Tape, Customer, XRay, and whatever other objects exist in your problem domain.

The ability to store non-native types directly in a database is one of the reasons why object oriented databases are frequently used in CAD and computer aided software engineering (CASE) applications. Flattening a complex set of classes, which are used in these types of applications, and then moving data to and from relational tables can be a complex, code-intensive task.

### D.2.3.2. Support for Inheritance

A real-world inheritance hierarchy must be flattened through either generalisation or specialisation so that it can be used in a relational database. *Generalisation* means collecting all attributes of the subclasses into a table used to store all occurrences of any of the subclasses; *specialisation* means creating separate tables to implement the subclasses. In specialisation, the attributes of the superclass are normally duplicated in each of the tables, based on the subclasses. This method of setting up an inheritance hierarchy is artificial; it is also unnecessary when using an object oriented database. Object oriented databases allow you to store objects directly that are subclasses or superclasses and then manipulate them without having to reconstruct them by selecting their attributes from more than one relational table.

### D.2.3.3. Performance

Most object oriented database implementations include a mechanism for rapid retrieval of an object based on its OID. Because of this, an application built using an object oriented database will frequently out-perform the same application built using a relational database. The performance benefits of an object oriented database can be especially dramatic when working with a large database.

## D.2.4. Disadvantages of Object Oriented Databases

It is important to note that object oriented databases are not appropriate in all circumstances. There are significant advantages to using a relational database. In the context of software tools, there are more and better tools for managing relational databases than their object oriented counterparts. Most object oriented databases are supported solely by their vendors. On the other hand, there is a large market of third-

party products for use with relational databases. These include tools for performing online backup, performance tuning and monitoring, database configuration and management.

From a management perspective it is important to take in to consideration the level of maintenance that the developed database will require. That is, there far more developers who have relational database experience than those with object oriented experience. In addition, experience with one relational database can easily be transferred to another.

At the time of writing there is little standardisation of object oriented databases. It would be difficult to take an application designed around one object oriented database and convert it to work with another. However, because a relational database is not so tightly integrated into an application's source code, it is easier to replace.

### D.2.5. Object Oriented or Relational Database?

In context of choosing between object oriented and relational databases there is no clear-cut rule about which database to use under which circumstances. However, as discussed earlier, object oriented databases have some benefits in certain areas, such as CAD and CASE applications. However, a relational database may be a better choice for an application in which the database lends itself particularly well to a table structure. In addition, the application developed might need to share data with an existing relational database. In this case, using a relational database for the new part of the system makes sense, to avoid the confusion of supporting both database models in a single system.

In summary, there are advantages to each type of database system. The decision relating to which system to choose should be made based on the particular goals and constraints of the system being developed.

Turning attention to the ICES case base component, it was initially envisaged that the database for storing cases would be object oriented. An object oriented database would enable the frame-based approach to data storage to be utilised, as outlined in Chapter 4. It was considered that the frame-based approach encapsulated within an object oriented database lent itself much more readily to the storage of aircraft case data than the traditional relational database format i.e., data relating to aircraft does not readily lend itself to a table-based format.

Having identified the database methodology which it was intended to use on a theoretical basis, it was necessary to examine the practicalities of implementing an object oriented database with C++ Builder. It quickly became apparent that it would not be possible to interface an existing proprietary object oriented database with C++ Builder. There are two principle reasons why this interface was not possible. Firstly, C++ Builder is a visual software tool and as such any third party database capability which was interfaced with it would also have to possess a compatible visual capability. After considerable research, no visual proprietary third party database was unearthed. It is important to note here, that unlike a relational database which is kept separate from the application, an object oriented database is embedded within the application. As such, it is essential that the object oriented database possess visual components or be able to interface with visual software. POET was the most promising of the object oriented databases examined but this package was very much geared towards interfacing with conventional C++.[103] The second reason why it was

not possible to interface an object oriented database with C++ Builder is that C++ Builder itself is geared towards interfacing with relational databases only. The database component classes which were discussed in section D.1.1.2 are designed to enable the user to interface with third party proprietary relational databases and are not flexible enough in their design to permit an interface with an object oriented database.

Having acknowledged that it was not possible to use an object oriented database with the ICES prototype but rather a relational database, it was necessary to identify the most suitable database for the task. C++ Builder comes with two standard drivers installed permitting an interface to dBase and Paradox. These two database packages appeared to offer the most likely database solution with respect to coding the case base component. When comparing these two software packages the most important feature to be considered was their respective ability to handle graphics. This is because a significant portion of aircraft data is best presented to the user in pictorial form. For example, it was considered essential that the user be presented with a graphic when discussing a component's structural configuration. An additional feature of relational database packages is that they are largely very similar i.e., the functionality in one relational database package will tend to be present in most others. With this in mind it was decided to first investigate the functionality of Paradox version 7 as this is a Borland product and as such it was considered likely that it would interface well with C++ Builder. The examination of Paradox version 7 revealed that the product interfaced seamlessly with C++ Builder and had excellent graphical display capabilities. As Paradox version 7 provided all the necessary functionality required with respect to developing the case base component of the ICES prototype it was considered unnecessary to investigate any other relational database alternative.

# Appendix E

# Research Project Timetable

# Research Project Plan